МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
Савченко А.С.

"____"_____20__ р.

ДИПЛОМНИЙ ПРОЕКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СПУПЕНЯ «БАКАЛАВР»

Тема: **«База даних обліку студентів та абітурієнтів ВНЗ»**

Виконавець: Форостяний Євген Олегович

Керівник: к.т.н.доцент. Моденов Ю.Б.

Нормоконтролер: Шевченко О.П.

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп‘ютерної та програмної інженерії

Кафедра комп’ютерних інформаційних технологій

Освітній ступінь: Бакалавр

Галузь знань, спеціальність, спеціалізація: 12 “Інформаційні технології”, 122 “Комп’ютерні науки ”, “Інформаційні управляючі системи та технології”

ЗАТВЕРДЖУЮ
Завідувач кафедри
А.С. Савченко
“_____”_____ 2021 р.

ЗАВДАННЯ
на виконання дипломного проекту студента
Форостяний Євген Олегович

1. Тема дипломного проекту: «База даних обліку студентів та абітурієнтів ВНЗ» затверджена наказом ректора від «22» квітня 2021 р. № 636/ст.

2. Термін виконання роботи: з 10.05.2021 до 20.06.2021;

3. Вихідні дані до роботи: На базі сучасних інформаційних технологій та засобів обчислювальної техніки спроектувати базу даних обліку студентів та абітурієнтів ВНЗ за замовленням начальника відділу інформаційно-технічного забезпечення;

4. Узгодження переліку запитів до бази даних, запрограмувати перелік запитів до реляційної бази даних на сучасних мовах запитів;

5. Зміст пояснювальної записки: Огляд і використання засобів СУБД для вирішення завдання дипломного проекту;

6. Перелік обов'язкового графічного матеріалу: Інфологічна модель предметної області, реляційна модель бази даних, перелік запитів до бази даних узгоджених із замовником, програми запитів до бази даних.

КАЛЕНДАРНИЙ ПЛАН

№ п/п	*Етапи виконання дипломної роботи*	*Термін виконання етапів*	*Примітка*
1	Вивчення літературних джерел	10.05.2021 – 11.05.2021	Виконав
2	Опис змісту предметної області Відділу інформаційно-технічного забезпечення	11.05.2021 – 15.05.2021	Виконав
3	Формальний апарат вирішення задачі роботи	15.05.2021– 16.05.2021	Виконав
4	Алгоритм синтезу реляційної моделі бази даних для Відділу інформаційно-технічного забезпечення	16.05.2021– 25.05.2021	Виконав
5	Проектування реляційної бази даних	25.05.2021– 30.05.2021	Виконав
6	Оформлення пояснювальної записки та графічної частини проекту	30.05.2021– 10.06.2021	Виконав

Керівник дипломного проекту: Моденов Ю.Б.

Студент: Форостяний Є.О.

РЕФЕРАТ

Дипломний проект «База даних обліку студентів та абітурієнтів ВНЗ» складається з вступу, 3 розділів: основної частини, висновків, списку літератури, додатку і містить 50 сторінок тексту, 12 рисунків.

Метою роботи є проектування бази даних для відділу інформаційно-технічного забезпечення, яка призначена для зберігання та використання даних про студентів університету та інших навчальних закладів, які підпорядковуються Національному авіаційному університету.

У роботі були використані наступні методи: метод «сутність-зв`язок», метод нормальних форм, сучасні підходи до вибору СУБД, підходи до проектування баз даних, метод перетворення інфологічної моделі в реляційну модель бази даних, складені програми запитів на мові SQL.

Практична цінність роботи полягає у можливості впровадження її результатів для відділу інформаційно-технічного забезпечення управління з персоналу та документозабезпечення Національного авіаційного університету.

Результати дипломного проектування рекомендується використовувати в практичній діяльності фахівців які забезпечують навчальний склад студентськими квитками.

АБІТУРІЄНТ, СТУДЕНТ, ПРЕДМЕТНА ОБЛАСТЬ, МОВА SQL, ІНФОЛОГІЧНА МОДЕЛЬ, БАЗА ДАНИХ, РЕЛЯЦІЙНА БАЗА ДАНИХ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

РБД	- реляційна база даних
БД	- база даних
СУБД	- система управління базами даних
ІС	- інформаційні системи
ІПС	- інформаційно-пошукові системи
ІТ	- інформаційні технології
Н.С.П.	- Нове Системне Проектування
ПО	- предметна область
СКК	- система класифікації і кодування
ПЗ	- програмне забезпечення
НФ	- нормальна форма
1НФ	- перша нормальна форма
2НФ	- друга нормальна форма
3НФ	- третя нормальна форма
БКНФ	- нормальна форма Бойса-Кода
4НФ	- четверта нормальна форма

ВСТУП

В сучасному світі будь-яка державна установа або приватна корпорація має бути забезпечена структурованим те безпечним сховищем даних. Одним з таких сховищ і є база даних розроблена виключно для виконання необхідних задач для забезпечення швидкодії та зручності корегування інформації.

Важливою перевагою використання бази даних в інформаційних системах є забезпечення незалежності даних від програм. Це дозволяє користувачам уникати вирішення проблем представлення даних на фізичному рівні: зберігання даних у пам'яті, спосіб доступу до них тощо.

У будь-якій організації, великій чи малій, існує проблема такої організації управління даними, яка забезпечить найбільш ефективну роботу. Деякі організації використовують картотеки, але більшість віддають перевагу комп'ютеризованим базам даних, які дозволяють їм ефективно зберігати та отримувати необхідну інформацію, а також керувати великими обсягами даних.

Саме тому темою даного дипломного проекту було обрано «База даних обліку студентів та абітурієнтів Вищого Навчального Закладу (ВНЗ)».

Метою дипломної роботи є створення і використання бази даних, в яких зберігається постійно обновлювана, максимально деталізована і систематизована по різноманітних ознаках інформація про студентів та абітурієнтів ВНЗ. Це дозволить оперативно відслідковувати наявність виготовлених студентських квитків, виготовлення їх дублікатів, мати повну інформацію про всіх студентів університету, їх переміщення з факультету на факультет, відрахування чи поновлення студентів на факультетах інститутів.

РОЗДІЛ 1
СУЧАСНА КОНЦЕПЦІЯ СТВОРЕННЯ БАЗИ ДАНИХ

1.1. Підходи до проектування баз даних

Широко відомий підхід до проектування баз даних (БД) з’явився в процесі розробки більш складних інформаційних систем, які повинні враховувати потреби не лише одного користувача, а великих груп та команд. Ці інтегровані бази даних призначені для вирішення широкого кола проблем, кожна з яких використовує лише одну частину даних, зазвичай перетинаються з іншими частинами, що використовуються в роботі. Тому основним підходом до проекту є усунення надлишків даних. Ці методи взаємодіють з іншими методами для забезпечення цілісності даних.

Встановлено основні вимоги до відокремлення програм від інтегрованих даних. Цей принцип спрямований на розділення даних як корпоративного ресурсу, він також важливий, оскільки характер збереження даних відокремлений від програми, яка часто змінюється.

Іншим важливим питанням проектування бази даних є те, що відомі необхідні експлуатаційні параметри, такі як кількість одиниць зовнішньої пам’яті або час проведення різних операцій та інші вимоги. Наприклад, дані не слід втрачати не лише через несправність пристрою, але й через помилку користувача. Це відрізняється від ситуації, коли вирішення проблем несе персональну відповідальність за збереження даних для цієї роботи.

Класична методологія проектування

Основний підхід до проектування баз даних - це потужний і красивий напрямок із філософією, способом розпізнавання реальності та її змісту. У цьому напрямку є прикладна математика, поняття "мир", "предмет" та їх приклади. Що стосується проектування бази даних, розуміється спосіб виконання наступних етапів проектування та інтегрується в систему координації:

- Збір інформації по предмету (аналіз вимог до програмного забезпечення та деталей із використанням методів, що називаються "процес" і вище, "перспектива застосування" та "непроцесорний підхід" або isp, "концепція структури інформації");
- Вибір обчислювальної мови, такий як "семантичний" формат для зберігання інформації про Тематичну область, аналіз та компіляція форматів баз даних;
- Аналіз даних, зібраних за Темою: класифікація, включення структурних елементів опису програмного забезпечення, ідентифікація як структурних, так і процедурних аспектів цілісності елементів у майбутній моделі програмного забезпечення, ідентифікація об'єктної моделі;
- Компіляція концептуальних моделей БД: інтегрована розробка БД проекту на вибраних мовах семантичного моделювання;
- Вибір конкретних форматів даних та баз даних для їх реалізації;
- Спроектувати логічну схему бази даних для обраної бази даних (що називається "дизайн реалізації");
- Розробка фізичної структури бази даних ("фізична" або "внутрішня", це - "форма"), включаючи розміщення бази даних на даних;
- Розробка технологій та процедури створення та заповнення бази даних;
- Розробка технологій та процедур ведення бази даних;
- Розробка міжнародної програми для доступу до БД та відповідних користувацьких інтерфейсів;
- Обробка даних для конкретних програм обробки даних: надання метаданих, контроль зразків даних тощо;

- Тестування бази даних, розробка та оновлення її структури.

Майстерня інструментів проектування БД

Розробка складних, стислих та організованих тем, великих баз даних стала непростим завданням. Наявність комплексного дизайнерського підходу дозволило нам доглядати за «дизайнером взуття» і почати шити його взуття у вигляді системи автоматизації проектування баз даних. Цьому сприяє технологічний досвід організації та компіляції систем розробки програмного забезпечення і, навпаки, використання інтегрованих словників даних. Таким чином, це відбувається в різних системах кейсів (автоматизовані інженерні системи) - системах структурного проектування баз даних та пов'язаних ІС, зосереджуючись на форматах даних, реалізованих у різних базах даних.

Таким чином, існує дві основні галузі для розробки систем кейсів та технологій проектування: кейси для спеціального проектування баз даних (або вдосконалених кейсів) та інтегровані інструменти, які дозволяють проектувати бази даних та розробляти додатки. Важливо зазначити, що в загальному просунуті випадки мають багато інструментів для пояснення функцій обробки даних (з використанням покрокового методу збору та аналізу даних за темами) та зберігання цих пояснень у бібліотеці. Це підтверджує місце несправного зв’язку між програмою бази даних та програмою ІР на основі цієї бази даних. Однак цей зв’язок не є досконалим, і принцип відокремлення БД від різних програм все ще існує.

Як правило, інтеграція сторінок дозволяє інтегрувати систему кейсів в єдину базу даних, яка фокусується на інструментах для розробки додатків. Такі комбінації висловлювались багато разів, наприклад, випадки зберігання були підкріплені "рідними" методами, але тільки СУБД, виробництво додатків виробляється "рідними" інструментами цієї розробки СУБД, і лише ними. Для таких інтегрованих систем випадків відображення концептуальних моделей баз даних у логічні діаграми часто робиться для конкретних баз даних.

Останній факт пов’язаний з іншим завданням, яке можна встановити для проектування БД: портативна конструкція БД, яка може бути реалізована на різних

комп’ютерних платформах, операційних системах, СУБД і навіть форматах даних і при необхідності передаватися з однієї платформи на іншу .

З огляду на це, семінар Classic Database Designer Workshop включає набір класичних методів структурного проектування, набір інструментів, придатних для створення прототипів, впровадження, завантаження та обслуговування. БД, а також «каскадну» організаційну схему виконання цих робіт по принципу «зверху вниз».

Тимчасові характеристики і транзакції

Забезпечення робочих характеристик бази даних все ще є непростим завданням, незважаючи на збільшення питомої потужності комп'ютера та зменшення вартості конкретної пам'яті. Тому визначення тимчасових характеристик роботи з БД та підтримання цих характеристик під час роботи БД належить до складного завдання проектування. У процесі проектування для визначення раціональної фізичної структури бази даних з методу визначення земних атрибутів потрібно наступне:

• Можливість порівняння тривалості впровадження різних систем баз даних у деяких СУБД;

• Можливість порівняння параметрів нових змін проектів БД в різних СУБД;

• Можливість порівняння параметрів реалізації однієї програми БД на різних апаратних серверах БД;

• Можливість прогнозувати тимчасові параметри програм та пристроїв.

Проблема порівняння часових шкал різних баз даних вважається самостійною. Однак це часто потрібно вирішувати як частину проектної роботи при виборі бази даних для бази даних, яку ми розробляємо, і в цьому процесі проектування.

Поняття транзакції визначено для визначення повного набору операцій у базі даних, який переводить дані з однієї бази даних, в розумному розумінні, до стану-кво. На його основі, насамперед, встановлюється механізм оновлення та відновлення правильної бази даних. Однак тоді на цій основі почали формуватися інші механізми та підходи.

Тимчасові оцінки СУБД найбільш популярних останнім часом тестів зумовлені кількістю транзакцій стандартного типу, встановлених на годину. Розподілена обробка базується на аутентифікації транзакцій.

Необхідно обмежити поділ праці. Тут ми відзначаємо дуже важливий ефект: практика зосередження уваги на «методах ведення бізнесу» тісно пов’язана з класичною моделлю. методологією проектування БД, яка розроблена насамперед як спосіб проектування баз даних, що називаються операційними базами даних, таких як бази даних, які повинні фіксувати дії кожного індивіда, який виконує, зберігаючи модель поточного фактичного стану об'єкта або простору.

Оцінка досягнутого стану

Можливо, ви, уявляєте, що ми маємо сучасний підхід або що ми збираємось його наблизити, але, на жаль, це не так, і, можливо, у нас ніколи не вийде. Не завжди складно мати підхід на рівні концепції, дуже важко застосувати на практиці. Каменем спотикання є труднощі проникнення в основний зміст теми (наприклад, складнощі в розумінні механізмів організації) та адаптація до бажаних умов роботи, можливо, навіть краща.

Подібна проблема характерна для всіх СУБД. Система баз даних повинна стати органічним компонентом системи управління організацією - це запорука успішного розгортання. Однак процес її впровадження передбачає деякі зміни в самій організації та в роботі її персоналу, і ми зіткнемося з неприродністю людей, коли мова зайде про сприйняття змін.

Дуже важливо, щоб інструмент СУБД відповідав потребам користувачів. Оскільки різним користувачам можуть знадобитися різні формати даних, мови та системи даних, доцільно, щоб база даних підтримувала декілька методів і щоб користувач міг вибрати найбільш підходящий.

Звичайно, можна поставити під сумнів цінність такого дослідження. Справді, незалежно від мови програмування, її все одно можна вивчити. Подібним чином інструменти СУБД можна час від часу модернізувати. Але проблема полягає не в розробці фондів, а в ефективності їх використання. Машини повинні служити людству, а не навпаки.

З тих пір СУБД, методи проектування баз даних та відповідні інструменти були значно додані. можливостях. Та решта світу теж не стояла на місці, суттєво ускладнилися завдання, що виконуються розробниками ІС та баз даних.

Обмеження класичних методів

Класичні форми та методи зосереджені на організації, зберіганні та обробці тонкоструктурованих даних, які відповідають поняттю "атрибути" як властивості об'єкта, представленого атомними елементами даних. З цієї причини, наприклад, повна база даних виділяється в розширену базу даних. Для їх проектування існує окремий напрямок - система пошуку інформації (IPS) або система пошуку інформації.

Через десятки років після оголошення свого класичного стилю та концепції класики чітко пояснили свої обмеження. Зазначено, що «під час обговорення моделювання даних акцент робиться на структурі, яка завдає шкоди обробці. Структурування без належних дій чи продумування методів подібне анатомії за відсутності науки.

Причини з`явлення нових вимог

Поява глобальних обчислювальних та персональних обчислень (разом із зменшенням вартості комп’ютерної периферії) призвело до багатьох нових можливостей для баз даних та їх проектування. Перелік типів даних, що збираються та обробляються, поширюється на можливі обмеження, визначені найпоширенішими стандартними значеннями "даної" концепції. База даних компанії включає не тільки відформатовані елементи та повні фрагменти тексту, але також базу даних з географічною інформацією, мультимедійну базу даних, і це не повний перелік.

Крім того, нові ІТ-можливості - поряд із чисто економічним обґрунтуванням - збільшили ринкові можливості та попит споживачів та призвели до значної конкуренції у галузях та послугах.. Ці підходи вимагають кардинальних змін в організації основних видів діяльності підприємства. Крім того: скорочення часу, кадрового забезпечення та інших витрат на виконання виробничих обов'язків.

Якщо ІТ є рушійною силою такого розвитку, вони будуть розроблені для забезпечення остаточного успіху та потенціалу підприємства. Існують вимоги до архітектури ІР компанії, як наслідок - уподобання до бази даних компанії.

Подібно до того, як сам ІС не можна розглядати ізольовано від своїх користувачів, амбіційний дизайн слід розглядати як поєднання трьох складових: умов вдосконалення бізнесу, людського фактора та підходу до ІТ. Справжнє поєднання цих

трьох компонентів, кожен з яких був придбаний у 1990-х роках, є якісним доповненням, створюючи вихідну точку того, що можна назвати системним дизайном.

Нові інструменти майстерні проектувальника, які ввійшли в практику

Мова sql, яка у 1980-х роках була лише половиною мови, що представляла модель взаємозв'язку, стала справжнім стандартом не лише для відповідних форматів даних, але і для галузевих баз даних. (Водночас це приклад придбання, яке може стати бідним)

У реальному розвитку організації та виробництві ІС, який є найбільш поширеним у більшості випадків або в більшості обсягів роботи, відбувається заміна методів розробки з sql, включаючи 3gl-мову або 4gl мову типу правил для мов Та інструменти 4gl до інтерфейсу інтерфейсу у форм-факторі та меню

Існує фактично сумісна робота з даними, насамперед - стандарт odbc.

Мультиплатформність стала стандартною, багатопротокольний зв’язок для розподілених баз даних реалізується на стандартизованій основі і контролює доступні транзакції, „міжнародний” був ретельно закодований з точки зору параметрів зашифрованих даних.

Структури та типи структур даних, впровадження структур реалізації даних у реалізацію: анонімні компоненти, повні бази даних та їх обробка, дані ГІС, мультимедійні бази даних, розподілені бази даних гіпертексту, обробка та обробка, розподілені разом із вхідними об’єктами ІР. На практиці існує кілька етапів, які сприяють справжній інтеграції цих структур та операцій.

Підхід до вибору СУБД змінюється, в першу чергу, щодо проектування, експлуатації та розробки баз даних організації, що планувалося щонайменше кілька років.

Визначення мети проектування баз даних тут не розглядається як новий інструмент для існуючих конференцій на практиці. (Об'єктно-орієнтоване програмування цього не означає.) В даний час представляється розумним прийняти такий дизайн у напрямку досліджень.

До нових підходів в організації проектування БД

Через збільшення кількості нових термінів, якщо вони не пов'язані зі збільшенням коефіцієнта конверсії термінів, новий підхід до проектування є більш актуальним для нової проектної організації.

Система всмоктування програмного забезпечення для управління всмоктуванням для IP вже давно перетворена на схему. Так, організація, що постійно розвивається, ibm corp. Забезпечує контрольований розвиток програмних систем у вигляді системних розподілів.

Що стосується проектування компонентів, організаційну таблицю проектування бази даних слід охарактеризувати як схему проектування паралельного потоку компонента бази даних та її складності за необхідності.

Часто кажуть, що об’єктно-орієнтоване проектування та програмування можуть вирішувати проблеми, що виникають внаслідок структурних підходів, і залишаються при використанні схемних програм. Однак із використанням таких технологій проблема семантичних взаємодій, особливо при проектуванні компонентів, насправді може ускладнюватися ще й кодуванням описів та акцентом на дисципліні документів даних. Поки що рано робити якісь висновки щодо обмежень використання цих методів.

Нові архітектурні принципи БД

Найважливішим завданням при розробці архітектури бази даних компанії є забезпечення різноманітних робочих процесів та ресурсів. Згідно з практикою, джерелами та користувачами інформації є не лише розподіл праці підприємства, штаб-квартира працівників або персонал міністерства, а й підприємства інших галузей (постачальники та потенційні споживачі, державні регулятори тощо). .).). Принципи глобалізації бізнесу визначають, що джерела даних та споживачі інформації мають географічне розташування там, де це потрібно.

Звідси і стратегічне рішення щодо архітектури бази даних та ІС в цілому. Поєднання вимог до динаміки та різноманітності типів інформаційних потоків, що експлуатуються в ІВ, з урахуванням зростання їх обсягу та необхідності

різноманітності методів обробки дає можливість загальним характеристикам технологій, що складають архітектуру бази даних:

Технологічні компоненти проектування та реконфігурації бази даних користувачів, що дозволяє здійснювати загальну роботу, включаючи, для зберігання даних, інтерфейси;

Удосконалена технологія зберігання даних, яка включає історичні дані, архіви, аудіо- та відеофайли, а також дані мультфільмів, що включає аналітичні методи обробки даних, основні типи зручних інтерфейсів, такі як гіпертекст, GeoInSet;

Відкрийте базу даних для компіляції та отримання даних з неї, використовуючи принципи глобальної швидкісної траси даних;

Архітектура відкритих систем, що розширюється за методами та дизайном компонентів: Найвищий рівень - це відкритість проектування компонентів БД та вільний обмін з будь-яким зовнішнім джерелом системи, на нижньому рівні - відкриття технології БД щодо стандартів мобільності, взаємодії, адаптації тощо.

Технологія Advanced Integrated Storage заснована на піднятті питання про розробку інтегрованого користувальницького інтерфейсу, що створить природні умови для роботи з даними та ресурсами незалежно від рівня збережених даних, який розробники сьогодні змушені призначати інформації (користувачеві). Розширена технологія Інтегрального Сховища змушує на новій основі порушувати питання про розробку інтегрованої сукупності інтерфейсів користувача, яка створювала б природні умови для роботи з інформацією й функціями незалежно від того, до якого класу збережених даних розроблювач змушений віднести сьогодні його (користувача) інформацію.

Виключення надмірності в даних

Залишається обґрунтованою вимогою до єдиного введення даних до бази даних для вирішення проблем та захисту від конфліктів (порушення логічної цілісності) під час оновлення збережених даних. Однак, з точки зору глобального інформаційного простору та дизайну компонентів, контекст цих положень потрібно переглянути. Безперечно, в оперативній базі даних доцільно планувати регулярні та

оптимістичні "острови", без пов'язаних груп або об'єктів. Більшість із цих «островів» надовго будуть базою даних відомих тем.

У той же час Історична інформаційна асоціація сховищ, БД ГІС-системи, збір текстових повідомлень, потік інформації, що надходить у інформаційний шлях, тощо в загальному звіті, дизайн бази даних компанії, загальне заперечення таємниці та примусу в базі даних.

Внутрішні ситуації, включаючи критичні та заздалегідь визначені потоки даних від даних магістралі даних до баз даних компанії, вимагатимуть розробки або вдосконалення "процедур ідентифікації" зразків структури даних, таких як визначення, чи описують ці приклади універсальну мету.

Консервація проблем

Справжній характер дизайнерської дисципліни, проілюстрований проектом мультфільму, методом структурного проектування та ієрархічною структурою та методологією, тепер штовхає дизайнерів звертатися до конкретних форм тексту. Технологічний випадок проектування баз даних повинен бути змінений таким чином, щоб виключити захист існуючих корпоративних проблем у суворій, "інтегрованій" структурі баз даних. Це може зажадати змін не тільки в технології, але і в інструментах проектування.

Метод призначення:

- Можливість фіксувати описи атрибутів, одиниць, взаємозв’язків, ролей тощо, які є дещо недосконалими, здатність описувати на рівні інформаційних наборів, які не підлягають пошуку та мають відношення до теми («група одиниць»);
- Розробити або побудувати прототипи IP та компонентів баз даних, їх інтеграція в загальні концепції;
- Ієрархічне проектування баз даних підприємств як послідовність баз даних, що працюють, включаючи: послідовні бази даних, бази даних, що визначають структуру «придбаних» робочих компонентів, розроблених спеціально для підприємств, цю базу даних, та бази даних двох останніх типів, послідовну та альтернативну IP;

- Відкритість сховищ системних випадків, словників СУБД та систем 4gl, які допомагають додавати метаоб'єкти та механізми з необхідним тезаурусом та глибокими семантичними зв'язками між елементами, а також робити двосторонній обмін метаінформацією з іншими системами 4gl і case, з`єднувати моделі різних компонентів в одну з використанням і збереженням усіх необхідних семантичних відношень.

Компонентна відкритість і смислова інтероперабельність

Компонентний підхід до розробки ІС вимагає проектування бази даних компонентів. Заміна деяких корисних компонентів IP подібною частиною, але розроблена іншим розробником, потребує заміни структури певної частини бази даних організації. Заміна повинна бути відтворена як новий етап проектування бази даних. При зміні компонентів бази даних інтерфейс існуючого додатка та їх користувачі повинні отримувати ті самі, з точки зору видимості, дані, що і раніше.

Фактичне проектування компонентів БД може базуватися на створенні та застосуванні загальних концептуальних моделей для складних компонентів та підтримці взаємозв'язку між моделями компонентів БД (та додатками, підключеними до них) та загальною концептуальною моделлю. Загалом офіційні вимоги до таких моделей були описані раніше (див. [Zinder90]). Нещодавно була розроблена повна формальна системна реалізація програмного забезпечення, заснована на об’єктно-орієнтованому підході, який забезпечує доступ до рішення цієї проблеми

Розробка понятійних моделей і СКК

Необхідність прийняття загальної концептуальної моделі змушує нас розглянути питання проектування бази даних так званих науково-дослідних інститутів (нормативні та довідкові дані) та СКК (класифікація та кодування).

До цього часу часто думали, що ССМ - це метод агрегування даних в інтегрованій базі даних. Насправді, відсутність СКК або використання неправильно сформованих СКК призведе до того, що семантичні недійсні дані зберігатимуться в іншій базі даних або навіть в одній базі даних. За цих умов це не призведе до досягнення цілей. Тому доцільно використовувати проектну роботу з БД Науково-дослідного та проектного інституту СКК як вихідну точку та як основу для створення

концептуального простору для теми, для побудови концептуальної моделі діяльності підприємства.

Понятійні моделі й наступні проектні роботи

На наступних етапах проектування власне БД понятійна модель продовжує використовуватися з різними цілями, наприклад:

- розробка сукупності різних предметних інформаційних моделей з виділенням загальних інформаційних сутностей;
- розробка функціональних моделей різних типів;
- розробка семантично багатих засобів підтримки користувача, і ін.

Технологічна відкритість

В ІС нової архітектури СУБД буде визначальним, але не єдиним компонентом інтегрованого програмного забезпечення (включаючи інтерфейс). Моніторинг бізнесу та процесів, семантичне моделювання та концептуалізація, СУБД - незалежний підхід до розробки та впровадження - інші класи програмних компонентів, що забезпечують досягнення цієї мети.

Рекомендується бути незалежним від СУБД на основі використання інструментів та стандартів, які охоплюють різні СУБД. Відмова у зв'язку зі стороною СУБД, відкритість зберігання справ, можливість розробки вкладених метамоделей у сховищах та застосований до них процес проектування є лише мінімальними вимогами до методів та інструментів.

Системи кейсів повинні зосереджуватися на можливості паралельного проектування елементів незалежними розробниками (в тому числі - без використання цієї системи випадків) з інтеграцією метаданих.

Засоби розробки розгортання повинні відповідати потребам мобільності програми та одночасно працювати в мінливому середовищі баз даних..

Проблеми обсягів, тимчасових характеристик і фізичного проектування

Поширення бази даних класів vldb вимагає використання декількох ефективних методів для створення ефективної фізичної діаграми даних. Неможливо створити таку базу даних на основі постійної реорганізації шляхом перепису в новій фізичній структурі. Це стосується операційних баз даних oltp, особливо олап-

орієнтованих терабайтних баз даних. Легкість процесу реорганізації цим методом може стати "пасткою" для дизайнерів, особливо - на ранніх стадіях розгортання бази даних, коли її перепис все ще можливий через неповний обсяг.

На рівні нових технологій (застосування декількох структур, індекси малих карт тощо) доцільно повернутися до прогнозуючого підходу експлуатаційних характеристик БД, що допоможе якнайменше планувати стабільність фізичних проектів без зовнішньоекономічні можливості. Для використання інших методів.

Значне збільшення обсягів баз даних супроводжуватиметься зростанням попиту на їх довіру. Через постійний процес проектування баз даних буде безпосередньо пов’язане з методами та прийомами проектування баз даних. Отже, для забезпечення відмовостійких даних необхідно мати методи контролю та координації з географічно розкиданими базами даних та резервними базами даних.

Проблема границь застосовності двох основних методів проектування

В ході досліджень та практичного проектування слід визначити обмеження застосування двох концепцій: дизайн БД як об'єкта, відокремленого від свідомості програми, та дизайн об'єкта, який інкапсулює як дані, так і метод його обробки.

Створення бази даних організації для проектування системи - це багатогранна діяльність, яка приймає багато форм класичного проектування, але вимагає іншого та більш організованого підходу, а також бази даних, що замінює те, що було розроблено 10 років тому.

Дисципліна проектування бази даних в нових умовах не змінилася. Однак видно його вихідну точку, її компоненти працюють у самому проекті. Відповідно до принципу захисту імунної системи від комп’ютерної корупції традиційні методи проектування баз даних повинні продовжувати застосовуватися, але лише в тих сферах, де вони справді корисні.

1.2. Процес створення бази даних

Весь процес створення бази даних в сучасному світі можна розділити на 5 основних кроків.

1.2.1. Збір вимог

Першим кроком є збір вимог. На цьому етапі розробники баз даних повинні опитати клієнтів (користувачів бази даних), щоб зрозуміти запропоновану систему та отримати та задокументувати дані та функціональні вимоги. Результатом цього кроку є документ, який включає докладні вимоги, надані користувачами.

Встановлення вимог передбачає консультації та узгодження між усіма користувачами щодо того, які постійні дані вони хочуть зберігати, а також домовленості щодо значення та інтерпретації елементів даних. Адміністратор даних відіграє ключову роль у цьому процесі, оскільки вони оглядають ділові, правові та етичні проблеми в організації, що впливають на вимоги до даних.

Документ вимог до даних використовується для підтвердження розуміння вимог користувачами. Щоб переконатися, що це легко зрозуміти, воно не повинно бути надмірно формальним чи кодованим. Документ повинен містити стислий виклад усіх вимог користувачів - а не лише сукупність вимог фізичних осіб - оскільки метою є створення єдиної спільної бази даних.

Вимоги не повинні описувати спосіб обробки даних, а саме те, що таке елементи даних, які атрибути вони мають, які обмеження застосовуються та взаємозв'язки між елементами даних.

1.2.2. Аналіз

Аналіз даних починається з викладу вимог до даних, а потім створюється концептуальна модель даних. Метою аналізу є отримання детального опису даних, який буде відповідати вимогам користувача з урахуванням властивостей даних високого та низького рівня, а також їх використання. Сюди входять такі властивості, як можливий діапазон значень, які можуть бути дозволені для атрибутів (наприклад, на прикладі шкільної бази даних, код курсу студента, назва курсу та кредитні бали).

Концептуальна модель даних забезпечує спільне формальне представлення того, що передається між клієнтами та розробниками під час розробки баз даних -

вона зосереджується на даних у базі даних, незалежно від можливого використання цих даних у процесах користувача або введення даних у конкретні IT-середовища . . Таким чином, концептуальна модель даних стосується значення та структури даних, але не деталей, що впливають на їх реалізацію.

Потім концептуальна модель даних - це офіційне представлення даних, які база даних повинна містити, та обмежень, які повинні поважати дані. Це слід виражати в термінах, які не залежать від того, як модель може бути реалізована. В результаті аналіз зосереджується на питаннях: "Що вам потрібно?" не "Як це робиться?"

1.2.3. Логічний дизайн

Розробка бази даних починається з концептуальної моделі даних і створює специфікацію логічної схеми; це визначить конкретний тип необхідної системи баз даних (мережева, реляційна, об’єктно-орієнтована). Реляційне представлення все ще не залежить від конкретної СУБД; це ще одна концептуальна модель даних.

Ми можемо використовувати реляційне представлення концептуальної моделі даних як вхідні дані для процесу логічного проектування. Результатом цього кроку є детальна реляційна специфікація, логічна схема всіх таблиць та обмежень, необхідних для задоволення опису даних у концептуальній моделі даних. Саме під час цієї проектної діяльності вибираються найбільш підходящі таблиці для подання даних у базі даних. Цей вибір повинен враховувати різні критерії проектування, включаючи, наприклад, гнучкість для змін, контроль дублікатів та те, як найкраще відображати обмеження. Саме таблиці, визначені логічною схемою, визначають, які дані зберігаються і як ними можна маніпулювати в базі даних.

Дизайнери баз даних, знайомі з реляційними базами даних та базами даних SQL, можуть спокуситися перейти безпосередньо до реалізації після створення концептуальної моделі даних. Однак таке безпосереднє перетворення реляційного подання в таблиці SQL не обов'язково призводить до створення бази даних, що має всі бажані властивості: повноту, цілісність, гнучкість, ефективність та простоту

використання. Хороша концептуальна модель даних є важливим першим кроком до бази даних з цими властивостями, але це не означає, що перетворення безпосередньо в таблицю SQL автоматично створює хорошу базу даних. На цьому першому кроці з точністю будуть представлені таблиці та обмеження, необхідні для задоволення опису концептуальної моделі даних, і, отже, відповідатиме вимогам повноти та цілісності, але він може бути жорстким або не дуже корисним. Тоді перший дизайн адаптується для поліпшення якості дизайну бази даних. Вигин - це термін, призначений охоплювати одночасні ідеї згинання чогось з іншою метою та послаблення його аспектів, коли воно згинається.

Рис. 1 узагальнює повторювані кроки, задіяні у розробці бази даних на основі наданого огляду. Його головна мета - відокремити загальне питання, які таблиці використовувати, від детального визначення компонентів кожної таблиці - ці таблиці розглядаються одна за одною, хоча вони не залежать одна від одної. Кожна ітерація, яка передбачає перегляд таблиць, призвела б до нового дизайну; в сукупності їх зазвичай називають вторинними структурами, хоча процес повторюється більше одного циклу.

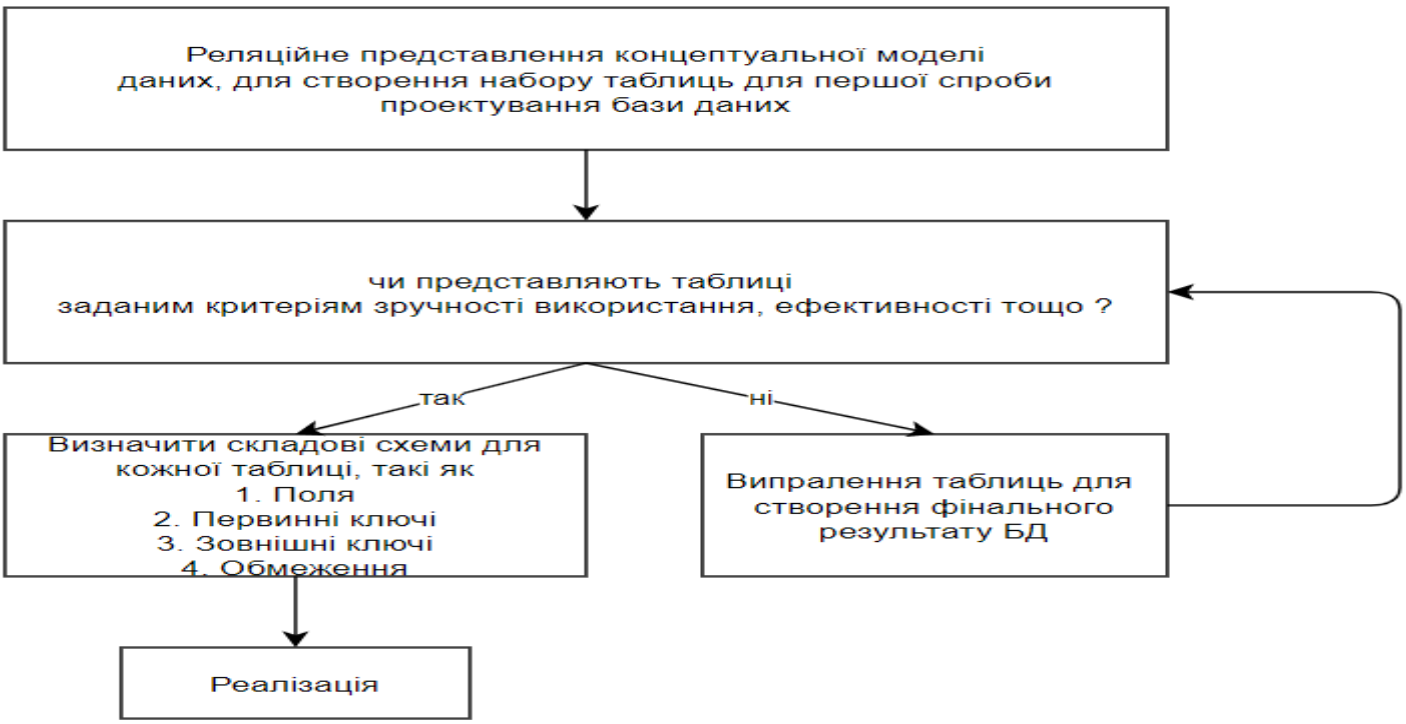


Рис. 1.1. Короткий зміст кроків, що беруть участь у розробці БД

По-перше, для даної концептуальної моделі даних не обов'язково, щоб усі вимоги користувачів, які вона представляє, задовольнялися єдиною базою даних. Можуть існувати різні причини для розробки декількох баз даних, такі як необхідність незалежної роботи в різних місцях або відомчий контроль над «своїми» даними. Однак, якщо колекція баз даних містить дубльовані дані, і користувачам потрібно отримати доступ до даних у декількох базах даних, то існують можливі причини, що одна база даних може задовольнити численні вимоги, або проблеми, пов'язані з реплікацією та розповсюдженням даних, повинні бути розглянуті.

По-друге, одне з припущень щодо розробки бази даних полягає в тому, що ми можемо відокремити розробку бази даних від розробки користувацьких процесів, які її використовують. Це базується на очікуванні, що після впровадження бази даних усі дані, необхідні поточно визначеним процесам користувача, будуть визначені та матимуть доступ до них; але ми також вимагаємо гнучкості, щоб дозволити нам відповідати майбутнім змінам вимог. Розробляючи базу даних для деяких додатків, можливо, можна передбачити загальні запити, які будуть подані до бази даних, і ми можемо оптимізувати наш дизайн для найбільш поширених запитів.

По-третє, на детальному рівні багато аспектів проектування та реалізації баз даних залежать від конкретної СУБД, що використовується. Якщо вибір СУБД фіксований або зроблений до завдання проектування, цей вибір може бути використаний для визначення критеріїв проектування, а не очікування до впровадження. Тобто, можна включати дизайнерські рішення для конкретної СУБД, а не створювати загальний дизайн, а потім пристосовувати його до СУБД під час реалізації.

Нерідко можна зустріти, що один дизайн не може одночасно задовольнити всі властивості хорошої бази даних. Тому важливо, щоб дизайнер визначив ці властивості пріоритетними (зазвичай використовуючи інформацію із специфікації вимог); наприклад, вирішити, чи цілісність важливіша за ефективність та чи корисність важливіша за гнучкість у даному розвитку.

В кінці етапу проектування логічна схема буде вказана за допомогою операторів мови визначення даних SQL (DDL), які описують базу даних, яку потрібно реалізувати для задоволення вимог користувача.

1.2.4. Реалізація

Реалізація передбачає побудову бази даних відповідно до специфікації логічної схеми. Це включатиме специфікацію відповідної схеми зберігання, забезпечення безпеки, зовнішню схему тощо. На реалізацію сильно впливає вибір доступних СУБД, інструментів баз даних та операційного середовища. Існують додаткові завдання, окрім простого створення схеми бази даних та впровадження обмежень - дані повинні вводитися в таблиці, вирішуватись питання, що стосуються користувачів та користувацьких процесів, а управлінські дії, пов’язані з більш широкими аспектами корпоративного управління даними, бути підтриманим. Відповідно до підходу до СУБД, ми хочемо, щоб якомога більше цих проблем було вирішено в рамках СУБД. Зараз ми коротко розглянемо деякі з цих проблем.

На практиці реалізація логічної схеми в певній СУБД вимагає дуже детальних знань про конкретні функції та засоби, які СУБД може запропонувати. В ідеальному світі, і відповідно до належної практики програмної інженерії, перший етап реалізації передбачав би узгодження вимог до дизайну з найкращими доступними інструментами реалізації, а потім використання цих інструментів для реалізації. Якщо говорити з точки зору бази даних, це може передбачати вибір продуктів постачальників із варіантами СУБД та SQL, які найбільш підходять для бази даних, яку нам потрібно впровадити. Однак ми живемо не в ідеальному світі, і частіше за все вибір обладнання та рішення щодо СУБД були прийняті задовго до розгляду проекту бази даних. Отже, реалізація може передбачати додаткове гнучкість дизайну для подолання будь-яких програмних чи апаратних обмежень.

Після створення логічного дизайну нам потрібно створити нашу базу даних відповідно до визначень, які ми створили. Для реалізації з реляційною СУБД це, ймовірно, передбачає використання SQL для створення таблиць та обмежень, які

задовольняють опис логічної схеми, та вибір відповідної схеми зберігання (якщо СУБД дозволяє такий рівень контролю).

Одним із способів досягти цього є запис відповідних операторів SQL DDL у файл, який може виконуватися СУБД, щоб існував незалежний запис, текстовий файл, операторів SQL, що визначають базу даних. Інший метод полягає в інтерактивній роботі за допомогою інструменту бази даних, такого як SQL Server Management Studio або Microsoft Access. Який би механізм не використовувався для реалізації логічної схеми, результатом є те, що база даних із таблицями та обмеженнями визначена, але не буде містити даних для процесів користувача.

1.2.5. Заповнення бази даних

Після створення бази даних існує два способи заповнення таблиць - або з наявних даних, або за допомогою користувацьких програм, розроблених для бази даних.

У деяких таблицях можуть бути наявні дані з іншої бази даних або файли даних. Наприклад, при створенні бази даних для лікарні, ви могли б очікувати, що вже є деякі записи всього персоналу, які повинні бути включені в базу даних. Дані можуть також надходити від зовнішнього агентства (списки адрес часто надходять від зовнішніх компаній) або створюватися під час великого завдання введення даних (перетворення паперових копій ручних записів у комп'ютерні файли може здійснюватися агентством введення даних). У таких ситуаціях найпростішим підходом для заповнення бази даних є використання засобів імпорту та експорту, які містяться в СУБД.

Зазвичай доступні засоби імпорту та експорту даних у різних стандартних форматах (ці функції також відомі в деяких системах як завантаження та розвантаження даних). Імпорт дозволяє копіювати файл даних безпосередньо в таблицю. Коли дані зберігаються у форматі файлу, який не підходить для використання функції імпорту, тоді необхідно підготувати прикладну програму, яка читає старі дані, перетворює їх за необхідності, а потім вставляє в базу даних за

допомогою спеціально створеного коду SQL для цього. Передача великої кількості існуючих даних до бази даних називається масовим завантаженням. Масове завантаження даних може містити дуже великі обсяги даних, що завантажуються, по одній таблиці за раз, тому ви можете виявити, що є засоби СУБД, які можуть відкласти перевірку обмежень до кінця масового завантаження.

1.3. Нормалізація бази даних

Нормалізація - це процес ефективної організації даних у базі даних. Існує дві цілі процесу нормалізації: усунення надлишкових даних (наприклад, зберігання одних і тих самих даних у більше ніж одній таблиці) та забезпечення сенсу залежності даних (лише зберігання пов'язаних даних у таблиці). І те, і інше є гідними цілями, оскільки вони зменшують обсяг місця, яке споживає база даних, і забезпечують логічне збереження даних.

Спільнота баз даних розробила ряд вказівок для забезпечення нормалізації баз даних. Вони називаються нормальними формами і нумеруються від однієї (найнижча форма нормалізації, яка називається першою нормальною формою або 1НФ), до п'яти (п'ята нормальна форма або 5НФ). У практичних додатках частіше зустрічаються 1НФ, 2ФН та 3ФН, а також іноді 4НФ. П’яту нормальну форму можна зустріти дуже рідко.

Перша нормальна форма (1НФ)

Перша нормальна форма (1НФ) встановлює основні правила для організованої бази даних:

- Виключіть повторювані стовпці з однієї таблиці.
- Створіть окремі таблиці для кожної групи пов’язаних даних та ідентифікуйте кожен рядок унікальним стовпцем або набором стовпців (первинний ключ).

Друга нормальна форма (2НФ)

Друга нормальна форма (2НФ) додатково розглядає концепцію видалення дублюючих даних:

- Відповідають усім вимогам першої нормальної форми.
- Видаліть підмножини даних, які застосовуються до декількох рядків таблиці, та розмістіть їх в окремих таблицях.
- Створіть взаємозв'язок між цими новими таблицями та їх попередниками за допомогою зовнішніх ключів.

Третя нормальна форма (3НФ

Третя нормальна форма (3НФ) робить один важливий крок далі:

- Відповідають усім вимогам другої нормальної форми.
- Видаліть стовпці, які не залежать від первинного ключа.

Звичайна форма Бойса Кода (БКНФ або 3.5НФ)

Звичайна форма Бойса-Кода, яку також називають "нормальною формою третьої та половини (3,5)", додає ще одну вимогу:

- Відповідають усім вимогам третьої нормальної форми.
- Кожен визначальний фактор повинен бути ключовим кандидатом.

1.4. Постановка задачі роботи

Отже, основна постанова задачі при створенні БД основується на:

- Створення інфологічної моделі;
- Перетворення інфологічної моделі в реляційну БД;
- Складення необхідних запитів та їх позгодження;
- Написання та перевірка запитів;

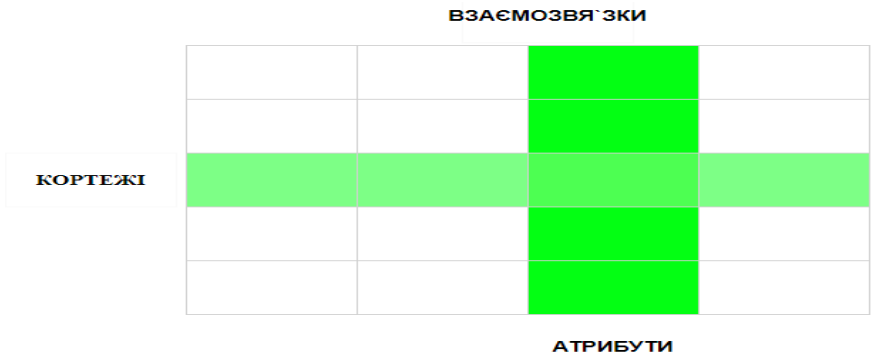
Висновок до розділу

В усьому описаному вище була описана сучасна концепція створення БД, розглянуто етапи створення та розроблена поставка задачі проектування бази даних обліку студентів та абітурієнтів ВНЗ.

РОЗДІЛ 2
ОГЛЯД ТА ПОРІВНЯННЯ ІСНУЮЧИХ СУБД

Бази даних - це пакети даних, змодельовані за допомогою логіки, тоді як система управління базами даних (СУБД) - це комп’ютерна програма, яка взаємодіє з цими базами даних. А СУБД використовується для запису даних, виконання запитів, контролю доступу та обробки інших завдань, пов'язаних з управлінням базами даних. Хоча терміни бази даних та системи управління базами даних використовуються як взаємозамінні, вони насправді не означають одне і те ж. База даних - це сукупність даних, яка може надходити з кількох джерел, а не просто зберігатись локально на комп’ютері. СУБД - це програмне забезпечення, яке дозволяє взаємодіяти з базою даних.

Кожна СУБД має структурну модель, яка диктує, як зберігати дані та отримувати до них доступ. Реляційна система управління базами даних (СУБД) використовує реляційну модель даних, в якій дані розбиваються на таблиці. У контексті СУБД вони називаються взаємозв'язками. Цей зв'язок складається з набору кортежів (рядків таблиці), кожен кортеж спільно використовує набір атрибутів (стовпців) з іншими рядками таблиці.



Кафедра КІТ				НАУ 21 44 98 000 УС			
Розробник	Форостяний Є.О.			ОГЛЯД ТА ПОРІВНЯННЯ ІСНУЮЧИХ СУБД	Літ.	Аркуш	Аркушів
Керівник	Моденов Ю.Б.					30	15
Консульт.					412 122		
Н. Контр.	Шевченко О.П.						
Зав.каф.	Савченко А.С.						

Основне програмне забезпечення СУБД діє як інтерфейс між кінцевим користувачем та базою даних, а також управляє даними, механізмами баз даних та системами баз даних для полегшення організації та маніпулювання даними.

Хоча функції СУБД дуже різні, але характеристики та можливості СУБД із загальним призначенням повинні включати елементи, які можуть отримувати доступ до описаного часу метаданих, бібліотеку управління системою СУБД, придушення інформації та незалежність, безпеку реєстрації даних та моніторинг діяльності підтримка валюти та переказ грошей, підтримка авторизованого доступу, доступ до підтримки з віддаленого боку, підтримка відновлення СУБД у випадку пошкодження та обмеження застосування, щоб забезпечити відповідність такої інформації певним правилам.

Методи проектування баз даних, призначені для підвищення чіткості організації даних, називаються звичайними явищами. Конвенції в СУБД модифікують існуючі шаблони, щоб зменшити дублювання та залежність даних, розбиваючи великі таблиці на менші таблиці та визначаючи взаємозв'язки між ними. Вихід СУБД - це набір вбудованих SQL в СУБД, що дозволяє користувачам відображати дані та результати та надсилати повідомлення з підпрограм, пакетів, блоків PL / SQL та тригерів. Спочатку Oracle розробив пакет передачі файлів СУБД, який включав кроки для копіювання двійкових файлів у базу даних або передачі двійкових файлів між базами даних.

Система управління базою даних працює за допомогою системних команд, спочатку отримуючи вказівки від адміністратора бази даних у СУБД, потім відповідно вказуючи системі, чи слід отримувати, змінювати чи завантажувати наявні дані з системи. Популярні приклади СУБД включають систему управління хмарними базами даних, систему управління базами даних пам'яті (IMDBMS), систему управління базами даних стовпців (CDBMS) та NoSQL у СУБД.

Реляційна система управління базами даних (СУБД) відноситься до набору програм та можливостей, призначених для того, щоб користувачі могли створювати, змінювати та керувати відповідними базами даних, які характеризуються логічними

структурами даних у незалежних таблицях. Існує кілька особливостей, що відрізняють реляційну СУБД від СУБД, зокрема:

- Структура: Якщо дані створюються в ієрархічному форматі в СУБД, дані структуруються у табличному форматі в реляційній СУБД.
- Можливості користувача: реляційна СУБД може працювати з кількома користувачами. СУБД може одночасно обробляти до одного користувача.
- Вимоги до програмного / апаратного забезпечення: реляційна СУБД має більше вимог до програмного та апаратного забезпечення.
- Керовані програми: СУБД підтримує базу даних у комп'ютерній мережі та на жорсткому диску системи. реляційна СУБД обробляє взаємозв'язки між таблицями зведених даних.
- Можливості даних: СУБД здатна обробляти невеликі обсяги даних, а реляційна СУБД може обробляти необмежений обсяг даних.
- Розподілена база даних: СУБД не забезпечує заміну розподілених баз даних, тоді як реляційна СУБД робить це.
- Впровадження ACID: модель реляційної СУБД базується на її структурі даних на моделі ACID (атомність, послідовність, ізоляція та довговічність).

Мова структурованих запитів (SQL)

Мова структурованих запитів (SQL) - це стандартизована мова для визначення та управління даними в реляційній базі даних. Відповідно до реляційної моделі даних база даних сприймається як набір таблиць, відносини представлені значеннями в таблицях, а дані отримуються шляхом вказівки таблиці результатів, яку можна отримати з однієї або декількох базових таблиць.

Інструкції SQL виконуються менеджером баз даних. Однією з функцій менеджера баз даних є перетворення специфікації таблиці результатів у послідовність внутрішніх операцій, що оптимізують пошук даних. Це перетворення відбувається під час підготовки оператора SQL. Ця трансформація також відома як зв'язування.

Усі виконувані оператори SQL повинні бути підготовлені перед їх виконанням. Результатом підготовки є виконувана або оперативна форма заяви.

Метод підготовки оператора SQL та збереження його робочої форми відрізняють статичний SQL від динамічного SQL.

Статичний SQL

Вихідна форма статичного оператора SQL вбудована в прикладну програму, написану мовою хосту, такою як COBOL, C або Java ™. Заява готується до виконання програми, а операційна форма заяви зберігається і після виконання програми.

Перед компіляцією вихідна програма, що містить статичні оператори SQL, повинна бути оброблена попереднім компілятором SQL. Прекомпілятор перевіряє синтаксис операторів SQL, перетворює їх у коментарі мови хосту та генерує оператори мови хосту для виклику менеджера баз даних.

Підготовка прикладної програми SQL включає попередню компіляцію, підготовку її статичних операторів SQL та компіляцію модифікованої вихідної програми.

Динамічний SQL

Програми, що містять вбудовані динамічні оператори SQL, повинні бути попередньо скомпільовані, як і ті, що містять статичний SQL, але на відміну від статичного SQL, динамічні оператори SQL створюються та готуються під час виконання. Вихідна форма заяви - це символ або графічний рядок, який передається програмі менеджеру баз даних за допомогою статичного оператора SQL PREPARE або EXECUTE IMMEDIATE. На оператор, підготовлений за допомогою оператора PREPARE, можна посилатися в операторі DECLARE CURSOR, DESCRIBE або EXECUTE. Оперативна форма заяви зберігається протягом усього періоду з'єднання або доти, поки остання програма SQL не покине стек викликів.

Інструкції SQL, вбудовані в додаток REXX, є динамічними операторами SQL. Оператори SQL, подані в інтерактивну систему SQL та інтерфейс рівня виклику (CLI), також є динамічними операторами SQL.

Розширений динамічний SQL

Розширений динамічний оператор SQL не є ні повністю статичним, ні повністю динамічним. API QSQPRCED надає користувачам розширені можливості динамічного SQL. Як і динамічний SQL, оператори можуть бути підготовлені,

описані та виконані за допомогою цього API. На відміну від динамічного SQL, оператори SQL, підготовлені в пакет за допомогою цього API, зберігаються до тих пір, поки пакет або оператор не буде явно відкинуто. Для отримання додаткової інформації див. Інформацію про API бази даних та файлів у категорії Програмування Інформаційного центру.

Інтерактивний SQL

Інтерактивна програма SQL пов'язана з кожним менеджером баз даних. По суті, кожен інтерактивний засіб SQL - це прикладна програма SQL, яка зчитує оператори з робочої станції, готує та виконує їх динамічно та відображає результати користувачеві. Кажуть, що такі оператори SQL видаються інтерактивно.

Інтерфейс рівня виклику SQL та відкрите підключення до бази даних

Інтерфейс рівня виклику DB2 (CLI) - це інтерфейс прикладного програмування, в якому програми надаються функції для обробки динамічних операторів SQL. DB2 CLI дозволяє користувачам будь-якої з мов отримувати доступ до функцій SQL безпосередньо через виклики процедур до сервісної програми, що надається DB2. Програми CLI також можна компілювати за допомогою набору розробників програмного забезпечення Open Database Connectivity (ODBC), доступного від Microsoft або інших постачальників, що забезпечує доступ до джерел даних ODBC. На відміну від використання вбудованого SQL, попередня компіляція не потрібна. Програми, розроблені з використанням цього інтерфейсу, можуть виконуватися на різних базах даних без компіляції щодо кожної з баз даних. За допомогою інтерфейсу програми використовують виклики процедур під час виконання для підключення до баз даних, видачі операторів SQL та отримання повернених даних та інформації про стан.

Інтерфейс CLI DB2 надає безліч функцій, недоступних у вбудованому SQL. Наприклад:

- CLI забезпечує виклики функцій, які підтримують послідовний спосіб запиту та отримання інформації про каталог систем баз даних у сімействі систем управління базами даних DB2. Це зменшує необхідність писати запити до каталогу для конкретного сервера додатків.

- Збережені процедури, викликані з прикладних програм, написаних за допомогою CLI, можуть повертати набори результатів до цих програм.

2.1. SQLite

SQLite - це реляційна СУБД із відкритим кодом, що базується на файлах, і є автономним. Він популярний завдяки своїй пристосованості та надійній роботі в середовищах з низьким обсягом пам’яті, а також портативності. Він використовує транзакції, які відповідають стандартам ACID, навіть коли система або виходить з ладу, або виникає відключення електроенергії.

SQLite описується як „безсерверна” база даних. Традиційні реляційні бази даних мають серверні процеси, що передбачають зв'язок програм із хост-сервером за допомогою запитів. Однак у SQLite будь-який процес, який взаємодіє з базою даних, читає з неї або записує безпосередньо у файл диска. Оскільки це позбавляє від потреби налаштовувати сервер, процес налаштування SQLite набагато простіший, ніж традиційна модель реляційних баз даних. Програми, що використовують SQLite, також не потребуватимуть додаткових налаштувань, оскільки їм просто потрібно керувати його доступом до диска.

SQLite не вимагає спеціального ліцензування, оскільки він є відкритим та безкоштовним. Проект пропонує деякі платні розширення з одноразовою платою за кожне. Вони допомагають у шифруванні та стисненні даних.

SQLite підтримує масив типів даних. Вони потрапляють у класи зберігання наступним чином:

ТИП ДАНИХ	ТЛУМАЧЕННЯ
NULL	ВКЛЮЧАЄ ВСІ NULL ЗНАЧЕННЯ
INTEGER	ВКЛЮЧАЄ ЦІЛІ ЧИСЛА В ДІАПАЗОНІ [-2147483648 до 2147483647]
REAL	ВКЛЮЧАЄ ВСІ ДІЙСНІ ЧИСЛА
TEXT	ТЕКСТОВІ РЯДКИ, ЩО ЗБЕРІГАЮТЬСЯ ЗА ДОПОМОГОЮ КОДУВАННЯ
BLOB	БУДЬ-ЯКІ ДАНІ ЩО ЗБЕРІГАЮТЬСЯ У ТАКОМУ Ж ФОРМАТІ ЯК І БУЛИ ВВЕДЕНІ

2.1.1. Переваги SQLite

Невеликий розмір: Оскільки SQLite є автономним, йому не потрібні зовнішні компоненти, які потребують інсталяції у вашій системі, щоб функціонувати. Використання простору SQLite (як випливає з назви) мінімальне. Хоча це дещо залежить від системи, загалом воно займає менше 600 КБ місця.

Зручність в користуванні: Іноді, що називається базою даних з нульовою конфігурацією, SQLite, по суті, готовий до використання під час встановлення. Оскільки SQLite не працює на стороні сервера, його ніколи не потрібно запускати, перезапускати або зупиняти. Також немає файлів конфігурації, якими вам потрібно керувати. Це робить SQLite простим для встановлення та налаштування інтеграції з програмою.

Мобільність: вся база даних SQLite знаходиться в одному файлі. На відміну від цього, інші системи управління базами даних зазвичай зберігають дані великими партіями окремих файлів. Місце розташування єдиного SQLite може вільно знаходитись де завгодно в ієрархії каталогів, і його можна перемістити за допомогою протоколу передачі та знімних носіїв

2.1.2. Недоліки SQLite

Обмеження паралельності: До бази даних SQLite доступні кілька підключень. Однак відбутися одночасно може лише одне редагування даних. Хоча SQLite і підтримує примітивні паралелізми, однак, він не має такої ж універсальності, як MySQL, PostgreSQL або інші СУБД клієнт / сервер.

Відсутність управління користувачами: Дозволи на доступ до SQLite - це типові обмеження операційних систем. Тому SQLite не є ідеальним вибором для додатків, де потрібні користувачі з власними дозволами.

Низький рівень безпеки: помилки в додатках спричиняють набагато менший збиток для механізму баз даних, що використовує сервер, оскільки важче пошкодити сторону сервера з точки зору цілісності даних. Система клієнт-сервер має краще керований доступ, паралельність та детальне блокування, ніж наявна у безсерверній СУБД.

2.2. MySQL

З тих пір, як сайт рейтингу DB-Engines почав відстежувати популярність баз даних у 2012 році, MySQL є найпопулярнішим реляційним СУБД із відкритим кодом. MySQL працює на одних з найбільших веб-сайтів у світі, включаючи Netflix, Spotify, Twitter та Facebook. Завдяки великій кількості документації та величезним співтовариством розробників, а також безліччю Інтернет-ресурсів, пов’язаних із MySQL, початок роботи з ним є простим процесом.

Створений з урахуванням надійності та доцільності, MySQL не повністю відповідає “стандартам SQL”. У той же час розробники MySQL постійно прагнуть їх дотримуватися. Проте рівень дотримання, як правило, відстає. На відміну від прямого доступу до програм, що використовують SQLite, за допомогою MySQL ви можете отримати доступ до бази даних через окремий запит до серверу. Це сприяє кращому контролю з точки зору доступу до бази даних, оскільки між додатками та самою базою даних є сервер.

MySQL породив багато сторонніх додатків, а також інтегровані бібліотеки, які допомагають не тільки легше працювати, але й розширюють його функціональність. Деякі з найбільш популярних сторонніх інструментів включають HeidiSQL, phpMyAdmin та DBeaver.

Типи даних MySQL розділені та організовані за трьома різними категоріями, включаючи дату / час, рядок та числові типи.

Дата/час:

ТИП ДАНИХ	ТЛУМАЧЕННЯ
DATA	ДАТА ПРЕДСТАВЛЕНА У ВИГЛЯДІ YYYY-MM-DD (2021-01-01)
DATATIME	ДАТА ПРЕДСТАВЛЕНА У ВИГЛЯДІ YYYY-MM-DD HH:MM:SS (2021-01-01 00:00:00)
TIMESTAMP	ПРЕДСТАЛЕННЯ ДАТИ ЯК К-СТЬ СЕКУНД ВІД ДАТИ ПОЧАТКУ ЕПОХИ UNIX(1970-01-01 00:00:00)
TIME	ЧАС ПРЕДСТАВЛЕНИЙ У ВИГЛЯДІ HH:MM:SS (00:00:00)
YEAR	ПРЕДСТАВЛЕННЯ РОКУ У ВИГЛЯДІ 2 АБО 4 ЦИФР (20 АБО 2020)

Рядки :

ТИП ДАНИХ	ТЛУМАЧЕННЯ
CHAR	РЯДОК ФІКСОВАНОЇ ДОВЖИНИ, ПРИ ЗБЕРІГАННІ ЗАВЖДИ ДОПОВНЮЄТЬСЯ ПРОБІЛАМИ В КІНЦІ РЯДКА ДО ЗАДАНОГО РОЗМІРУ
VARCHAR	РЯДОК ЗМІННОЇ ДОВЖИНИ
BLOB	БУДЬ-ЯКІ ДАНІ, ЩО ЗБЕРІГАЮТЬСЯ У ТАКОМУ Ж ФОРМАТІ ЯК І БУЛИ ВВЕДЕНІ
TEXT	ТЕКСТОВІ РЯДКИ,ЩО ЗБЕРІГАЮТЬСЯ ЗА ДОПОМОГОЮ КОДУВАННЯ
TINYBLOB, TINYTEXT	СТОВБЕЦЬ BLOB АБО TEXT З МАКСИМАЛЬНОЮ ДОВЖИНОЮ 255 (2 ^ 8 - 1) СИМВОЛІВ.
MEDIUMBLOB, MEDIUMTEXT	СТОВБЕЦЬ BLOB АБО TEXT З МАКСИМАЛЬНОЮ ДОВЖИНОЮ 16777215 (2 ^ 24 - 1) СИМВОЛІВ.
LONGBLOB, LONGTEXT	СТОВБЕЦЬ BLOB АБО TEXT З МАКСИМАЛЬНОЮ ДОВЖИНОЮ 4294967295 (2 ^ 32 - 1) СИМВОЛІВ.

Числові типи:

ТИП ДАНИХ	ТЛУМАЧЕННЯ
TINYINT	ЦІЛЕ ЧИСЛО В ДІАПАЗОНІ ЗІ ЗНАКОМ ВІД -128 ДО 127. БЕЗ ЗНАКА ВІД 0 ДО 255.
SMALLINT	ЦІЛЕ ЧИСЛО. В ДІАПАЗОНІ ЗІ ЗНАКОМ ВІД -32768 ДО 32767. БЕЗ ЗНАКА ВІД 0 ДО 65535
MEDIUMINT	ЦІЛЕ ЧИСЛО В ДІАПАЗОНІ ЗІ ЗНАКОМ ВІД -8388608 ДО 8388607. БЕЗ ЗНАКА ВІД 0 ДО 16777215
INT(INTEGER)	ЦІЛЕ ЧИСЛО В ДІАПАЗОНІ ЗІ ЗНАКОМ ВІД -2147483648 ДО 2147483647.БЕЗ ЗНАКА ВІД 0 ДО 4294967295
BIGINT	ЦІЛЕ ЧИСЛО В ДІАПАЗОНІ ЗІ ЗНАКОМ ВІД -9223372036854775808 ДО 9223372036854775807. БЕЗ ЗНАКА ВІД 0 ДО 18446744073709551615
FLOAT	ДРОБОВІ ЧИСЛА В ДІАПАЗОНІ ВІД -3.4028 * 1038 ДО 3.4028 * 1038
DOUBLE	ДРОБОВІ ЧИСЛА В ДІАПАЗОНІ ВІД -1.7976 * 10308 ДО1.7976 * 10308
BOOL(BOOLEAN)	МОЖЕ ЗБЕРІГАТИ ЛИШЕ 2 ЗНАЧЕННЯ TRUE АБО FALSE

2.2.1. Переваги MySQL

Зручність у використанні та популярність: Завдяки своїй нестримній популярності MySQL не тільки має велику кількість Інтернет-матеріалів та друкованої документації. Крім цього, MySQL має кілька додаткових інструментів, таких як phpMyAdmin, які допомагають спростити процес початку роботи.

Безпека: MySQL включає сценарій, пропонований під час встановлення, який допомагає поліпшити захист бази даних, визначаючи пароль кореневого користувача, видаляючи ті бази даних, які доступні всім користувачам за замовчуванням, та встановлюючи рівні захисту паролем. Крім того, на відміну від SQLite, MySQL дозволяє контролювати параметри доступу до прав користувача.

Швидкість роботи: Розробники MySQL змогли визначити пріоритети швидкості завдяки своєму вибору не реалізовувати всі функції SQL. Хоча нещодавно інші системи управління базами даних набули успіху, MySQL з точки зору швидкості роботи, він все ще має репутацію одного з найшвидших СУБД.

Реплікація бази даних: MySQL підтримує багаторазову реплікацію. Реплікація сприяє доступності бази даних, надійності та стійкості до несправностей шляхом обміну інформацією між кількома хостами.

2.2.2. Недоліки MySQL

Чіткі обмеження: Оскільки MySQL розроблений для зручності використання та доцільності, існують деякі функціональні обмеження через відсутність дотримання вимог SQL. Прикладом є підтримка речень FULL JOIN.

Повільний розвиток: з моменту придбання MySQL у 2008 році Sun Microsystems та наступного 2009 року корпорацією Oracle, користувачі відзначають більш млявий процес розробки, оскільки спільнота з відкритим кодом більше не реагує на виправлення проблем та розробку.

2.3. PostgreSQL

Також відомий як Postgres, PostgreSQL вважає себе найдосконалішою серед існуючих реляційних баз даних. Створений з метою сумісності стандартів та розширень, PostgreSQL, об'єктно-реляційна СУБД, представляє такі функції, як успадкування таблиці та перевантаження функцій. Ці функції найчастіше пов’язані з об’єктними базами даних.

PostgreSQL має надзвичайно бажану функцію паралельності (ефективно обробляє кілька завдань одночасно). Це досягається впровадженням мультиверсійного контролю паралельності (MVCC), що забезпечує відповідність ACID (атомність, послідовність, ізоляція та довговічність) транзакцій.

Хоча PostgreSQL не настільки популярний, як MySQL, все ще існує безліч сторонніх інструментів, що полегшують роботу з ним. Деякі приклади таких інструментів - Postbird та pgAdmin.

Подібно до MySQL, PostgreSQL підтримує числові типи, дату / час та рядкові типи даних. Однак він також підтримує інші типи даних, включаючи мережеві адреси, геометричні фігури, бітові рядки, записи JSON, пошук тексту та інші ідіосинкратичні типи даних.

Числові типи:

ТИП ДАНИХ	ТЛУМАЧЕННЯ
SMALLINT	ЦІЛЕ ЧИСЛО. В ДІАПАЗОНІ ВІД -32768 ДО 32767
INT(INTEGER)	ЦІЛЕ ЧИСЛО В ДІАПАЗОНІ ВІД -2147483648 ДО 2147483647
BIGINT	ЦІЛЕ ЧИСЛО В ДІАПАЗОНІ ВІД -9223372036854775808 ДО 9223372036854775807
SMALLSERIAL	ЦІЛЕ ЧИСЛО, ЩО АВТОМАТИЧНО ЗБІЛЬШУЄТЬСЯ В ДІАПАЗОНІ ВІД 1 ДО 32767
SERIAL	ЦІЛЕ ЧИСЛО, ЩО АВТОМАТИЧНО ЗБІЛЬШУЄТЬСЯ В ДІАПАЗОНІ ВІД 1 ДО 2147483647
BIGSERIAL	ЦІЛЕ ЧИСЛО, ЩО АВТОМАТИЧНО ЗБІЛЬШУЄТЬСЯ В ДІАПАЗОНІ ВІД 1 ДО 9223372036854775807
DOUBLE PRECISION	ДРОБОВІ ЧИСЛА В ДІАПАЗОНІ ВІД -1.7976 * 10308 ДО1.7976 * 10308

Дата/час:

ТИП ДАНИХ	ТЛУМАЧЕННЯ
DATE	ДАТА
INTERVAL	ЧАСОВИЙ ІНТЕРВАЛ
TIME	ЧАС
TIME WITH TIMEZONE	ЧАС З УРАХУВАННЯМ ЧАСОВОГО ПОЯСУ
TIMESTAMP	ДАТА ТА ЧАС ПРЕДСТАВЛЕНІ ЯК К-СТЬ СЕКУНД ВІД 01-01-1970
TIMESTAMP WITH TIMEZONE	ДАТА ТА ЧАС ПРЕДСТАВЛЕНІ ЯК К-СТЬ СЕКУНД ВІД 01-01-1970 З УРАХУВАННЯМ ЧАСОВОГО ПОЯСУ

Геометричні типи:

ТИП ДАНИХ	ТЛУМАЧЕННЯ
POINT	ТОЧКА НА ПЛОЩИНІ (X;Y;)
LINE	ПРЯМА {A;B;C}
LSEG	ВІДРІЗОК ((X1;Y1)(X2;Y2))
BOX	ПРЯМОКУТНИК (X1;Y1....)
POLYGON	БАГАТОКУТНИК (X1;Y1....)
CIRCLE	КОЛО <(X;Y)R>

Інші типи:

ТИП ДАНИХ	ТЛУМАЧЕННЯ
JSON	ТЕКСТОВІ ДАНІ В ФОРМАТІ JSON
BOOLEAN	МОЖЕ ЗБЕРІГАТИ ЛИШЕ 2 ЗНАЧЕННЯ TRUE АБО FALSE
MONEY	СУМА В ВАЛЮТІ
XML	ДАНІ В ФОРМАТІ XML
CIDR	IPV4\IPV6 АДРЕС МЕРЕЖІ

2.3.1. Переваги PostgreSQL

Відповідність SQL: PostgreSQL прагне досягти відповідності SQL набагато більше, ніж MySQL або SQLite. Він підтримує 160 із 179 функцій, необхідних для виконання базового стандарту SQL: 2011. Він також містить багато додаткових функцій згідно офіційної документації.

Спільнота та відкритий код: PostgreSQL має величезну та дуже віддану спільноту розробників, яка підтримує та надає безліч Інтернет-ресурсів, які описують, як найкраще використовувати її. Сюди входять різні форуми в Інтернеті, офіційна документація та навіть Wiki-програма PostgreSQL.

Розширюваність: Використовуючи динамічне завантаження та керування каталогом, користувачі можуть програмно розширювати PostgreSQL на льоту. Файл об’єктного коду може бути призначений (наприклад, спільна бібліотека) для завантаження, якщо потрібно, PostgreSQL.

2.3.2. Недоліки PostgreSQL

Проблеми з продуктивністю пам'яті: Кожне нове підключення клієнта додає новий процес у PostgreSQL. Кожен процес займає приблизно 10 Мб пам'яті, виділеної для нього. Це може призвести до великої кількості користувацьких з'єднань, які займають великі обсяги пам'яті. Це означає, що PostgreSQL працює не так добре з важкими середовищами для читання, як інші СУБД.

Популярність: Незважаючи на те, що останні роки продемонстрували високий рівень використання, з точки зору популярності, PostgreSQL в значній мірі відставав від MySQL. Одна з причин полягає в тому, що у Postgre доступно менше сторонніх інструментів. Крім цього, просто менше адміністраторів баз даних із досвідом роботи з базами Postgre порівняно з MySQL.

Висновок до розділу

В другому розділі були розглянуті 3 найбільш популярні системи управління базами даних, а також проведене їх порівняння, визначення переваг та недоліків кожного. Після їх аналізу, для реалізації бази даних студентів та абітурієнтів Вищого Навчального Закладу було обрано реляційну СУБД MySQL, так як вона цілком відповідає всім необхідним критеріям розробки, має надзвичайно велику кількість ресурсів, що допомагають розробникам, а також є однією з найбільш безпечних СУБД.

РОЗДІЛ 3
РЕЛЯЦІЙНА МОДЕЛЬ БД ОБЛІКУ СТУДЕНТІВ ТА АБІТУРІЄНТІВ ВНЗ

3.1. Опис змісту предметної області

База даних призначена для зберігання та використання даних про студентів та абітурієнтів Вищих Навчальних Закладів.

Отже, виготовлення студентських квитків для абітурієнтів виконується у такому порядку: спочатку абітурієнт вступаючи до навчального закладу подає документи до приймальної комісії. В приймальній комісії кожного факультету є відповідальний виконавець, який надає потрібні дані для виготовлення студентського квитка до відповідального секретаря приймальної комісії. Відповідальний виконавець призначений деканатом свого факультету. В свою чергу, відповідальний секретар приймальної комісії, після зарахування абітурієнта до навчального закладу, подає накази про зарахування та данні на студента першого курсу до відділу ІТЗ для виготовлення студентських квитків терміном на чотири роки. Спочатку надається наказ про зарахування абітурієнтів з Інституту довузівської підготовки, потім студентів які зараховані на бюджет, а вже потім студентів, які зараховані на контракт.

У випадку якщо студентський квиток був загублений, студентом пишеться заява з проханням відновити студентський квиток, підписується в деканаті та за допомогою відповідальної особи передається до відділу ІТЗ. У випадку зміни прізвища студентом пишеться заява з проханням замінити студентський квиток, додаючи відповідні документи.

Предметна область має такі сутності:

1. Абітурієнт;
2. Студент.

В свою чергу кожна з сутностей має такі атрибути:

1.Абітурієнт

1.1. Номер по порядку;

1.2. Прізвище;

1.3. Ім`я;

1.4. По батькові;

1.5. Стать (чоловік, жінка);

1.6. Дата народження;

1.7. Тип документу (свідоцтво про народження, паспорт);

1.8. Серія документу;

1.9. Номер документу;

1.10. Фінансування (бюджет, контракт);

1.11. Ідентифікаційний код;

1.12. Відзнака або пільга для студентського квитка (без відзнаки, з відзнакою (золота медаль), срібна медаль, без пільги, пільговий);

1.13. Дата навчання (початок);

1.14. Дата навчання (закінчення);

1.15. Назва файлу фотокартки;

1.16. Форма навчання (стаціонар, заочна, вечірня);

1.17. Дата видачі студентського квитка;

1.18. Дата закінчення терміну студентського квитка;

1.19. Назва факультету;

1.20. Назва навчального закладу (університету, інституту, коледжу, технікуму, ліцею);

1.21. Код напряму спеціальності;

1.22. Назва напряму спеціальності;

1.23. Курс;

1.24. Вид студентського квитка (оригінал, дублікат);

1.25. Громадянство;

1.26. Серія студентського квитка;

1.27. Номер студентського квитка.

2. Студент:

2.1. Номер по порядку;

2.2. Прізвище;

2.3. Ім`я;

2.4. По батькові;

2.5. Стать (чоловік, жінка);

2.6. Дата народження;

2.7. Тип документу (свідоцтво про народження, паспорт);

2.8. Серія документу;

2.9. Номер документу;

2.10. Фінансування (бюджет, контракт);

2.11. Ідентифікаційний код;

2.12. Відзнака або пільга для студентського квитка (без відзнаки, з відзнакою (золота медаль), срібна медаль, без пільги, пільговий);

2.13. Дата навчання (початок);

2.14. Дата навчання (закінчення);

2.15. Назва файлу фотокартки;

2.16. Форма навчання (стаціонар, заочна, вечірня);

2.17. Дата видачі студентського квитка;

2.18. Дата закінчення терміну студентського квитка;

2.19. Назва факультету;

2.20. Група;

2.21. Назва навчального закладу;

2.22. Код спеціальності;

2.23. Назва спеціальності;

2.24. Курс;

2.25. Вид студентського квитка (оригінал, дублікат);

2.26. Громадянство;

2.27. Серія студентського квитка;

2.28. Номер студентського квитка.

3.2. Реляційна модель бази даних

За узгодженням із замовником БД розбита на дві частини:

1. Абітурієнт (рис.3.1);
2. Студент (рис.3.2)

Це рішення прийнято для зручності роботи.

База даних була створена на базі СУБД MySQL за допомогою додатку PHPMyAdmin.

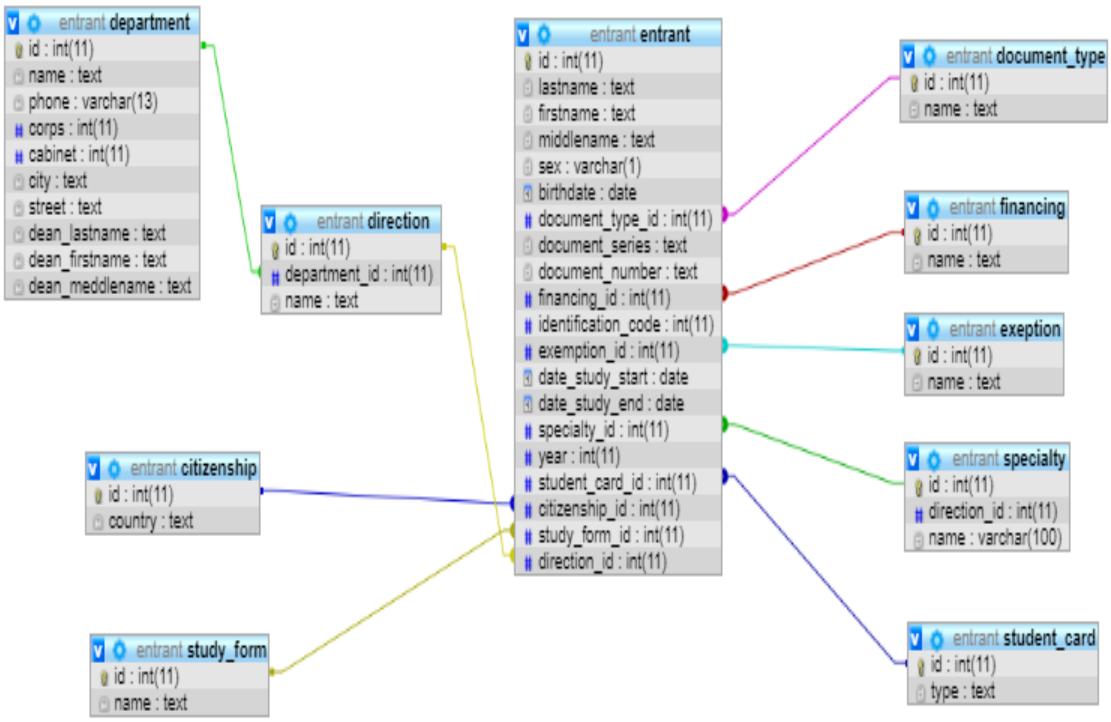


Рис. 3.1 Абітурієнт

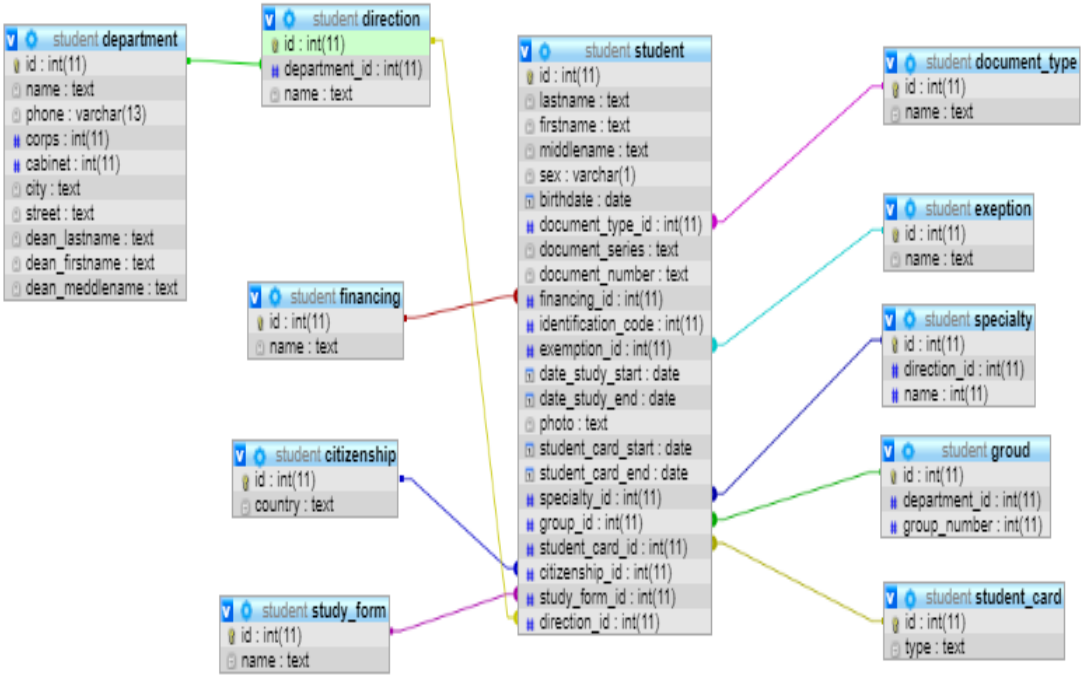


Рис. 3.2. Студент

3.3. Запити до спроектованої бази даних

1. Скільки всього абітурієнтів вступило на перший курс.
2. Скільки абітурієнтів вступило на перший курс на контракт.
3. Скільки абітурієнтів вступило на перший курс на бюджет.
4. Скільки студентів всього в університеті.
5. Скільки всього студентів денної форми навчання.
6. Скільки всього студентів заочної форми навчання.
7. Скільки іноземних студентів в університеті.
8. Скільки іноземних студентів заочної форми навчання.
9. Скільки всього студентів забезпечено студентськими квитками.
10. Скільки всього іноземних студентів забезпечено студентськими квитками по країнам.
11. Скільки іноземних студентів денної форми навчання забезпечено студентськими квитками.

12. Скільки всього українських студентів забезпечено студентськими квитками.
13. Скільки українських студентів денної форми навчання забезпечено студентськими квитками.
14. Скільки українських студентів заочної форми навчання забезпечено студентськими квитками.
15. Скільки всього студентів не забезпечено студентськими квитками.
16. Кількість студентів не забезпечених студентськими квитками по факультетам, групам.
17. Кількість студентів університету заочної форми навчання по курсам.
18. Скільки студентів контрактників навчається на факультеті
19. Скільки студентів бюджетників навчається на факультеті.

3.4. Програма запитів мовою SQL

Запит №1

```
SELECT Count(id)
FROM entrant.entrant
HAVING (((Count(entrant.year))=1));
```

Запит №2

```
SELECT entrant.entrant.year, Count(entrant.financing.name)
FROM entrant.financing
INNER JOIN entrant.entrant ON entrant.financing_id=entrant.financing.id
GROUP BY entrant.entrant.year
HAVING (((entrant.entrant.year)=1) AND
((Count(entrant.financing.name))='контракт'));
```

Запит №3

```
SELECT entrant.entrant.year, Count(entrant.financing.name)
FROM entrant.financing
INNER JOIN entrant.entrant ON entrant.financing_id=entrant.financing.id
```

```
GROUP BY entrant.entrant.year
HAVING (((entrant.entrant.year)=1) AND
((Count(entrant.financing.name))='бюджет'));
```

Запит №4

```
SELECT Count(student.id)
FROM student.student;
```

Запит №5

```
SELECT student.study_form.name, Count(student.student.id)
FROM student.study_form INNER JOIN student.student ON student.study_form.id =
student.study_form_id
GROUP BY student.study_form.name
HAVING student.study_form.name='денна';
```

Запит №6

```
SELECT student.study_form.name, Count(student.student.id)
FROM student.study_form INNER JOIN student.student ON student.study_form.id =
student.study_form_id
GROUP BY student.study_form.name
HAVING student.study_form.name='заочна';
```

Запит №7

```
SELECT Count(student.student.id), student.citizenship.country
FROM student.citizenship INNER JOIN student.student ON student.student.citizenship_id
= student.citizenship.id
GROUP BY student.citizenship.country
HAVING student.citizenship.country<>'Україна';
```

Запит №8

```
SELECT Count(student.student.id), student.citizenship.country,student.study_form.name
FROM student.citizenship INNER JOIN student.student ON student.student.citizenship_id
= student.citizenship.id
INNER JOIN student.study_form ON
```

student.study_form.id=student.student.study_form_id

GROUP BY student.citizenship.country,student.study_form.name

HAVING student.citizenship.country<>'Україна' and student.study_form.name='заочна';

Запит №9

SELECT Count(student.student.id), student.student_card.type

FROM student.student_card INNER JOIN student.student ON student.student_card.id = student.student.student_card_id

GROUP BY student.student_card.type

HAVING student.student_card.type<>'відсутні';

Запит №10

SELECT Count(student.student.id), student.student_card.type,student.citizenship.country

FROM student.student_card INNER JOIN student.student ON student.student_card.id = student.student.student_card_id

INNER JOIN student.citizenship ON student.student.citizenship_id = student.citizenship.id

GROUP BY student.student_card.type AND student.citizenship.country

HAVING student.student_card.type<>'відсутні' AND

student.citizenship.country<>'Україна';

Запит №11

SELECT Count(student.student.id), student.student_card.type,student.citizenship.country

FROM student.student_card INNER JOIN student.student ON student.student_card.id = student.student.student_card_id

INNER JOIN student.citizenship ON student.student.citizenship_id = student.citizenship.id

INNER JOIN student.student ON student.study_form.id = student.study_form_id

GROUP BY student.student_card.type AND student.citizenship.country

HAVING student.student_card.type<>'відсутні' AND

student.citizenship.country<>'Україна' and student.study_form.name='денна';

Запит №12

SELECT Count(student.student.id), student.student_card.type,student.citizenship.country

FROM student.student_card INNER JOIN student.student ON student.student_card.id = student.student.student_card_id

```
INNER JOIN student.citizenship ON student.student.citizenship_id = student.citizenship.id
GROUP BY student.student_card.type AND student.citizenship.country
HAVING student.student_card.type<>'відсутні' AND
student.citizenship.country='Україна';
```

Запит №13

```
SELECT Count(student.student.id), student.student_card.type,student.citizenship.country
FROM student.student_card INNER JOIN student.student ON student.student_card.id =
student.student.student_card_id
INNER JOIN student.citizenship ON student.student.citizenship_id = student.citizenship.id
INNER JOIN student.student ON student.study_form.id = student.study_form_id
GROUP BY student.student_card.type AND student.citizenship.country
HAVING student.student_card.type<>'відсутні' AND
student.citizenship.country='Україна' and student.study_form.name='денна';
```

Запит №14

```
SELECT Count(student.student.id), student.student_card.type,student.citizenship.country
FROM student.student_card INNER JOIN student.student ON student.student_card.id =
student.student.student_card_id
INNER JOIN student.citizenship ON student.student.citizenship_id = student.citizenship.id
INNER JOIN student.student ON student.study_form.id = student.study_form_id
GROUP BY student.student_card.type AND student.citizenship.country
HAVING student.student_card.type<>'відсутні' AND
student.citizenship.country='Україна' and student.study_form.name=’заочна’;
```

Запит №15

```
SELECT Count(student.student.id), student.student_card.type
FROM student.student_card INNER JOIN student.student ON student.student_card.id =
student.student.student_card_id
GROUP BY student.student_card.type
HAVING student.student_card.type='відсутні';
```

Запит №16

```
SELECT Count(student.student.id), student.group.group_number,
student.department.name, student.student_card.type
FROM student.department
INNER JOIN student.`group` ON student.`group`.department_id=student.department.id
INNER JOIN student.student ON student.student.group_id=student.`group`.id
INNER JOIN student.student_card ON
student.student.student_card_id=student.student_card.id
GROUP BY student.`group`.id, student.department.name, student.student_card.type
HAVING student.student_card.type='відсутні';
```

Запит №17

```
SELECT Count(student.student.id), student.study_form.name, student.student.year
FROM student.study_form INNER JOIN student.student ON student.study_form.id =
student.student.study_form_id
GROUP BY student.study_form.id and student.student.year
HAVING student.study_form.name='заочна';
```

Запит №18

```
SELECT Count(student.student.id), student.study_form.name, student.department.name
FROM student.student
INNER JOIN student.`group` ON student.student.group_id=student.`group`.id
INNER JOIN student.department ON student.department.id =
student.`group`.department_id
INNER JOIN student.study_form ON
student.student.study_form_id=student.study_form.id
GROUP BY student.study_form.name,student.department.name
HAVING student.study_form.name='контракт';
```

Запит №19

```
SELECT Count(student.student.id), student.study_form.name, student.department.name
FROM student.student
INNER JOIN student.`group` ON student.student.group_id=student.`group`.id
```

```
INNER JOIN student.department ON student.department.id =
student.`group`.department_id
INNER JOIN student.study_form ON
student.student.study_form_id=student.study_form.id
GROUP BY student.study_form.name,student.department.name
HAVING student.study_form.name='бюджет';
```

ВИСНОВКИ

В результаті виконання дипломного проекту було досліджено сучасну концепцію створення баз даних, проведено ретельний аналіз підходів до проектування БД та розроблено алгоритм створення бази даних. Був проведений огляд та порівняння найбільш популярних СУБД, під час якого мій вибір зупинився на MySQL, так як, ця СУБД повністю задовольняла критерії розробки бази даних на обрану тему, а також мала в своєму арсеналі додатки за допомогою яких процес створення БД був значно зручнішим. Після цього розпочався етап проектування та розробки “Бази даних обліку студентів та абітурієнтів ВНЗ”, а саме:

- Проведено аналіз вимог для подальшого створення концептуальної моделі;
- Визначено основні сутності та атрибути БД;
- Створений логічний дизайн та визначено необхідний тип бази даних(Реляційна БД);
- Реалізована база даних на основі MySQL;
- Створена та виконана програма запитів на мові SQL.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Інформаційно-пошукові системи. [Електронний ресурс] – Режим доступу: http://esu.com.ua/ (дата звернення 28.04.21р) – Назва з екрана.
2. Інформаційно-пошукові системи. [Електронний ресурс] – Режим доступу: http://articles.org.ru/ (дата звернення 01.05.21р) – Назва з екрана.
3. Критерии выбора СУБД при создании информационных систем. [Електронний ресурс] – Режим доступу: http://www.interface.ru/home.asp?artId=2147 (дата звернення 05.05.21р) – Назва з екрана.
4. СУБД Postgres Pro. [Електронний ресурс] – Режим доступу: https://postgrespro.ru/ (дата звернення 10.05.21р) – Назва з екрана.
5. About SQLite. [Електронний ресурс] – Режим доступу: https://www.sqlite.org/index.html (дата звернення 10.05.21р) – Назва з екрана.
6. MySQL. [Електронний ресурс] – Режим доступу: https://www.mysql.com/ (дата звернення 10.05.21р) – Назва з екрана.
7. Пасічник В. В. Організація баз даних та знань / В. В. Пасічник, В. А. Резніченко. – К.: Видавнича група BHV, 2006. – 384 с..
8. Пономаренко В. С. Інформаційні системи в управлінні персоналом.. Навчальний посібник / В. С. Пономаренко, І. В. Журавльова, І. Л. Латишева. – Харків: Вид. ХНЕУ, 2008. – 336 с. (Укр. мов.) Електрон. аналог друк. вид.: режим доступу: http://www.repository.hneu.edu.ua/ (дата звернення 18.04.2021 р). – Назва з екрана.
9. Хомоненко А. Д. Базы данных: Учебник для высших учебных заведений. Навчальний посібник / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев - Санкт-Петербург: Вид КОРОНА-Век, 2009. - 736 с. - Електрон. аналог друк. вид.: режим доступу: https://studizba.com/ (дата звернення 01.05.2021 р). – Назва з екрана.

ДОДАТКИ

Додаток А

Створення Бази даних «Абітурієнт»:

База даних: `entrant`

```
CREATE TABLE `citizenship` (
  `id` int(11) NOT NULL,
  `country` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE `department` (
  `id` int(11) NOT NULL,
  `name` text,
  `phone` varchar(13) NOT NULL,
  `corps` int(11) NOT NULL,
  `cabinet` int(11) NOT NULL,
  `city` text NOT NULL,
  `street` text NOT NULL,
  `dean_lastname` text,
  `dean_firstname` text,
  `dean_meddlename` text
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE `direction` (
  `id` int(11) NOT NULL,
  `department_id` int(11) NOT NULL,
  `name` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE `document_type` (
  `id` int(11) NOT NULL,
  `name` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE `entrant` (
```

```
  `id` int(11) NOT NULL,
  `lastname` text NOT NULL,
  `firstname` text NOT NULL,
  `middlename` text NOT NULL,
  `sex` varchar(1) NOT NULL,
  `birthdate` date NOT NULL,
  `document_type_id` int(11) NOT NULL,
  `document_series` text NOT NULL,
  `document_number` text NOT NULL,
  `financing_id` int(11) NOT NULL,
  `identification_code` int(11) NOT NULL,
  `exemption_id` int(11) NOT NULL,
  `date_study_start` date NOT NULL,
  `date_study_end` date NOT NULL,
  `specialty_id` int(11) NOT NULL,
  `year` int(11) NOT NULL,
  `student_card_id` int(11) NOT NULL,
  `citizenship_id` int(11) NOT NULL,
  `study_form_id` int(11) NOT NULL,
  `direction_id` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE `exeption` (
  `id` int(11) NOT NULL,
  `name` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE `financing` (
  `id` int(11) NOT NULL,
  `name` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE `specialty` (
  `id` int(11) NOT NULL,
  `direction_id` int(11) NOT NULL,
  `name` varchar(100) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE `student_card` (
  `id` int(11) NOT NULL,
  `type` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE `study_form` (
  `id` int(11) NOT NULL,
  `name` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
ALTER TABLE `citizenship`
  ADD PRIMARY KEY (`id`);
ALTER TABLE `department`
  ADD PRIMARY KEY (`id`),
  ADD KEY `phone` (`phone`),
  ADD KEY `corps` (`corps`);
ALTER TABLE `direction`
  ADD PRIMARY KEY (`id`),
  ADD KEY `direction_department_id_fk` (`department_id`);
ALTER TABLE `document_type`
  ADD PRIMARY KEY (`id`);
ALTER TABLE `entrant`
  ADD PRIMARY KEY (`id`),
  ADD KEY `document_type_id` (`document_type_id`),
  ADD KEY `financing_id` (`financing_id`),
  ADD KEY `exemption_id` (`exemption_id`),
```

```
  ADD KEY `citizenship_id` (`citizenship_id`),
  ADD KEY `student_card_id` (`student_card_id`),
  ADD KEY `group_id` (`year`),
  ADD KEY `entrant_specialty_id_fk` (`specialty_id`),
  ADD KEY `entrant_study_form_id_fk` (`study_form_id`),
  ADD KEY `entrant_direction_id_fk` (`direction_id`);
ALTER TABLE `exeption`
  ADD PRIMARY KEY (`id`);
ALTER TABLE `financing`
  ADD PRIMARY KEY (`id`);
ALTER TABLE `specialty`
  ADD PRIMARY KEY (`id`),
  ADD KEY `direction_id` (`direction_id`);
ALTER TABLE `student_card`
  ADD PRIMARY KEY (`id`);
ALTER TABLE `study_form`
  ADD PRIMARY KEY (`id`);
ALTER TABLE `citizenship`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `department`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `document_type`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `entrant`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `exeption`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `financing`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
```

```
ALTER TABLE `student_card`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `study_form`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `direction`
  ADD CONSTRAINT `direction_department_id_fk` FOREIGN KEY (`department_id`)
REFERENCES `department` (`id`);
ALTER TABLE `entrant`
  ADD CONSTRAINT `entrant_citizenship_id_fk` FOREIGN KEY (`citizenship_id`)
REFERENCES `citizenship` (`id`),
  ADD CONSTRAINT `entrant_direction_id_fk` FOREIGN KEY (`direction_id`)
REFERENCES `direction` (`id`),
  ADD CONSTRAINT `entrant_document_type_fk` FOREIGN KEY (`document_type_id`)
REFERENCES `document_type` (`id`),
  ADD CONSTRAINT `entrant_exemption_id_fk` FOREIGN KEY (`exemption_id`)
REFERENCES `exeption` (`id`),
  ADD CONSTRAINT `entrant_financing_id_fk` FOREIGN KEY (`financing_id`)
REFERENCES `financing` (`id`),
  ADD CONSTRAINT `entrant_specialty_id_fk` FOREIGN KEY (`specialty_id`)
REFERENCES `specialty` (`id`),
  ADD CONSTRAINT `entrant_student_card_id_fk` FOREIGN KEY (`student_card_id`)
REFERENCES `student_card` (`id`),
  ADD CONSTRAINT `entrant_study_form_id_fk` FOREIGN KEY (`study_form_id`)
REFERENCES `study_form` (`id`);
COMMIT;
```

Створення Бази даних «Студент»:

```
База данных: `student`
CREATE TABLE `citizenship` (
  `id` int(11) NOT NULL,
```

```
  `country` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE `department` (
  `id` int(11) NOT NULL,
  `name` text,
  `phone` varchar(13) NOT NULL,
  `corps` int(11) NOT NULL,
  `cabinet` int(11) NOT NULL,
  `city` text NOT NULL,
  `street` text NOT NULL,
  `dean_lastname` text,
  `dean_firstname` text,
  `dean_meddlename` text
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE `direction` (
  `id` int(11) NOT NULL,
  `department_id` int(11) NOT NULL,
  `name` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE `document_type` (
  `id` int(11) NOT NULL,
  `name` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE `exeption` (
  `id` int(11) NOT NULL,
  `name` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE `financing` (
  `id` int(11) NOT NULL,
```

```
  `name` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE `group` (
  `id` int(11) NOT NULL,
  `department_id` int(11) NOT NULL,
  `group_number` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE `specialty` (
  `id` int(11) NOT NULL,
  `direction_id` int(11) NOT NULL,
  `name` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE `student` (
  `id` int(11) NOT NULL,
  `lastname` text NOT NULL,
  `firstname` text NOT NULL,
  `middlename` text NOT NULL,
  `sex` varchar(1) NOT NULL,
  `birthdate` date NOT NULL,
  `document_type_id` int(11) NOT NULL,
  `document_series` text NOT NULL,
  `document_number` text NOT NULL,
  `financing_id` int(11) NOT NULL,
  `identification_code` int(11) NOT NULL,
  `exemption_id` int(11) NOT NULL,
  `date_study_start` date NOT NULL,
  `date_study_end` date NOT NULL,
  `photo` text NOT NULL,
  `student_card_start` date NOT NULL,
```

```
  `student_card_end` date NOT NULL,
  `specialty_id` int(11) NOT NULL,
  `group_id` int(11) NOT NULL,
  `student_card_id` int(11) NOT NULL,
  `citizenship_id` int(11) NOT NULL,
  `study_form_id` int(11) NOT NULL,
  `direction_id` int(11) NOT NULL,
  `year` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE `student_card` (
  `id` int(11) NOT NULL,
  `type` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE `study_form` (
  `id` int(11) NOT NULL,
  `name` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
ALTER TABLE `citizenship`
  ADD PRIMARY KEY (`id`);
ALTER TABLE `department`
  ADD PRIMARY KEY (`id`),
  ADD KEY `phone` (`phone`),
  ADD KEY `corps` (`corps`);
ALTER TABLE `direction`
  ADD PRIMARY KEY (`id`),
  ADD KEY `student_department_id_fk` (`department_id`);
ALTER TABLE `document_type`
  ADD PRIMARY KEY (`id`);
ALTER TABLE `exeption`
```

```
  ADD PRIMARY KEY (`id`);
ALTER TABLE `financing`
  ADD PRIMARY KEY (`id`);
ALTER TABLE `group`
  ADD PRIMARY KEY (`id`),
  ADD KEY `department_id` (`department_id`),
  ADD KEY `group_number` (`group_number`);
ALTER TABLE `specialty`
  ADD PRIMARY KEY (`id`),
  ADD KEY `direction_id` (`direction_id`);
ALTER TABLE `student`
  ADD PRIMARY KEY (`id`),
  ADD KEY `document_type_id` (`document_type_id`),
  ADD KEY `financing_id` (`financing_id`),
  ADD KEY `exemption_id` (`exemption_id`),
  ADD KEY `citizenship_id` (`citizenship_id`),
  ADD KEY `student_card_id` (`student_card_id`),
  ADD KEY `group_id` (`group_id`),
  ADD KEY `student_specialty_id_fk` (`specialty_id`),
  ADD KEY `student_study_form_id_fk` (`study_form_id`),
  ADD KEY `student_direction_id_fk` (`direction_id`);
ALTER TABLE `student_card`
  ADD PRIMARY KEY (`id`);
ALTER TABLE `study_form`
  ADD PRIMARY KEY (`id`);
ALTER TABLE `citizenship`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `department`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
```

```
ALTER TABLE `document_type`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `exeption`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `financing`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `group`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `student`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `student_card`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `study_form`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `direction`
  ADD CONSTRAINT `student_department_id_fk` FOREIGN KEY (`department_id`)
REFERENCES `department` (`id`);
ALTER TABLE `student`
  ADD CONSTRAINT `student_citizenship_id_fk` FOREIGN KEY (`citizenship_id`)
REFERENCES `citizenship` (`id`),
  ADD CONSTRAINT `student_direction_id_fk` FOREIGN KEY (`direction_id`)
REFERENCES `direction` (`id`),
  ADD CONSTRAINT `student_document_type_id_fk` FOREIGN KEY
(`document_type_id`) REFERENCES `document_type` (`id`),
  ADD CONSTRAINT `student_exemption_id_fk` FOREIGN KEY (`exemption_id`)
REFERENCES `exeption` (`id`),
  ADD CONSTRAINT `student_financing_id_fk` FOREIGN KEY (`financing_id`)
REFERENCES `financing` (`id`),
  ADD CONSTRAINT `student_group_id_fk` FOREIGN KEY (`group_id`) REFERENCES
```

```
`group` (`id`),
  ADD CONSTRAINT `student_specialty_id_fk` FOREIGN KEY (`specialty_id`)
REFERENCES `specialty` (`id`),
  ADD CONSTRAINT `student_student_card_id_fk` FOREIGN KEY (`student_card_id`)
REFERENCES `student_card` (`id`),
  ADD CONSTRAINT `student_study_form_id_fk` FOREIGN KEY (`study_form_id`)
REFERENCES `study_form` (`id`);
COMMIT;
```